NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

APPLICATIONS OF NEURAL NETWORKS TO
ADAPTIVE CONTROL

by

Russell W. Scott II

December 1989

Thesis Advisor: Prof. D. J. Collins

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 31	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
Applications of Neural Networks to Adaptive Control

12 PERSONAL AUTHOR(S)
Scott, Russell W. II

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Engineer's Thesis	FROM _____ TO _____	December 1989	117

16 SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Neural Networks, Adaptive Control, Backpropagation, Parameter Estimation, Parallel Distributed Processing

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The amount of a priori knowledge required to design some modern control systems is becoming prohibitive. Two current methods addressing this problem are robust control, in which the control design is insensitive to errors in system knowledge, and adaptive control, in which the control law is adjusted in response to a continually updated model of the system. This thesis examines the application of parallel distributed processing (neural networks) to the problem of adaptive control. The structure of neural networks is introduced, focusing on the Backpropagation paradigm. A general form of controller consistent with use in neural networks is developed and combined with a discussion of linear least squares parameter estimation techniques to suggest a structure for a neural network adaptive controller. This neural network adaptive control structure is then applied to a number of estimation and control problems using as a model the longitudinal motion of the A-4 aircraft. The purpose of this thesis is to develop and demonstrate a neural network adaptive control structure consistent with adaptive control theory.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL Dr. D. J. Collins	22b TELEPHONE (Include Area Code) (408) 646-2826	22c OFFICE SYMBOL 67Co

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

Approved for Public release; distribution unlimited.

APPLICATIONS OF NEURAL NETWORKS TO ADAPTIVE CONTROL

by

Russell W. Scott II
Lieutenant, United States Navy
B.S.M.E., United States Naval Academy, 1981
M.S.A.E., Naval Postgraduate School, March 1989

Submitted in partial fulfillment of the
requirements for the degree of

AERONAUTICAL ENGINEER

from the

NAVAL POSTGRADUATE SCHOOL
December 1989

ABSTRACT

The amount of a priori knowledge required to design some modern control systems is becoming prohibitive. Two current methods addressing this problem are robust control, in which the control design is insensitive to errors in system knowledge, and adaptive control, in which the control law is adjusted in response to a continually updated model of the system. This thesis examines the application of parallel distributed processing (neural networks) to the problem of adaptive control. The structure of neural networks is introduced, focusing on the Backpropagation paradigm. A general form of controller consistent with use in neural networks is developed and combined with a discussion of linear least squares parameter estimation techniques to suggest a structure for neural network adaptive controllers. This neural network adaptive control structure is then applied to a number of estimation and control problems using as a model the longitudinal motion of the A-4 aircraft. The purpose of this thesis is to develop and demonstrate a neural network adaptive control structure consistent with adaptive control theory.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

I. INTRODUCTION 1

II. NEURAL NETWORK THEORY 3

A. NEURAL NETWORK PROCESSING 3

B. NEURAL NETWORK ARCHITECTURE 4

1. Processing Elements 4

2. State of Activation 6

3. Connections 6

4. Activation Rule 7

5. Propagation Rule 8

6. Learning Rule 8

7. Environment 8

C. NEURAL NETWORK OPERATION 9

D. BACKPROPAGATION ALGORITHM 9

1. Architecture 10

2. Backpropagation Learning Rule 10

3. The Backpropagation Activation Rule 13

4. The Power of the Backpropagation Neural Network 16

III. ADAPTIVE CONTROL THEORY 17

A. ONE STEP AHEAD PREDICTION CONTROL 17

B. LINEAR LEAST SQUARES ESTIMATION 20

C. NEURAL NETWORKS AND ADAPTIVE CONTROL 22

IV. NEURAL NETWORK ADAPTIVE CONTROL STRUCTURES 24

A. PARALLEL STRUCTURE 24

B. SEQUENTIAL STRUCTURE 26

V. EXPERIMENTAL SETUP 30
A. HARDWARE-SOFTWARE PACKAGE 30
B. LONGITUDINAL MOTION OF THE A-4 AIRCRAFT 32
C. EXPERIMENTAL DESIGN CONSIDERATIONS 37
1. Control Design Issues 37
2. Estimation Design Issues 39
a. Input-Output Selection 39
b. Model Selection 44
3. Validation Issues 48
D. SUMMARY OF EXPERIMENTAL SETUP 49
VI. RESULTS AND DISCUSSION 50
A. NEURAL NETWORK STABILITY CHARACTERISTICS 50
B. THE NEURAL NETWORK ADAPTIVE CONTROLLER IN ESTIMATION 56
1. Linear Neural Network Parameterized as Four Transfer Functions . . 56
2. Fully Connected Linear Neural Network 63
3. Nonlinear Neural Network Estimators 65
4. Nonlinear Network Modelling Linear System 66
5. Multiple Condition Nonlinear Neural Network Estimator 67
C. CONTROL USING THE NEURAL NETWORK ADAPTIVE CONTROLLER 70
D. SUMMARY 74
VII. CONCLUSIONS AND RECOMMENDATIONS 78
REFERENCES 81
APPENDIX A: NEURALWORKS PROFESSIONAL II ASSOCIATED PROGRAMS 82
APPENDIX B: MATLAB M-FILE 97

APPENDIX C: CONTINUOUS STATE SPACE EQUATIONS AND DISCRETE MATRIX POLYNOMIALS 98
INITIAL DISTRIBUTION LIST 103

LIST OF TABLES

Table I: Flight Conditions Selected for Study 34

Table II: Parameters for Flight Condition Sea Level/Mach 0.4 with a Sampling Time of 0.1 Seconds 37

Table III: Poles and Zeros of the Discrete Simulation for Condition 1 with a Sampling Time of 0.1 Seconds 38

Table IV: Network Weights at 5,000 and 5,000,000 Cycles 57

LIST OF FIGURES

Figure 1: A Typical Neural Network Structure 5

Figure 2: A Typical Processing Element 5

Figure 3: Neuralworks Professional II Activation Function Logic 7

Figure 4: Plot of the Sigmoid Activation Function 14

Figure 5: Plot of the Hyperbolic Tangent Activation Function 15

Figure 6: Adaptive Control Structure 18

Figure 7: Model Reference Adaptive Control Structure 23

Figure 8: Parallel Neural Network Adaptive Controller 25

Figure 9: Sequential Neural Network Adaptive Controller: Estimation Phase 26

Figure 10: Sequential Neural Network Adaptive Controller: Control Phase 27

Figure 11: A-4 Frequency Response 34

Figure 12: Frequency Response for Discrete A-4 Longitudinal Motion Simulation . . . 36

Figure 13: The Effect of Sampling Rate on Poles and Zeros 41

Figure 14: Effect of Sampling Rate on Excitation 42

Figure 15: Neural Network Adaptive Controller Structure for A-4 Longitudinal Motion 45

Figure 16: Non-linear Neural Network Adaptive Controller Structure for A-4 Longitudinal Motion/Multiple Conditions 46

Figure 17: Network Static Stability for u(t) 52

Figure 18: Network Static Stability for $\alpha(t)$ 52

Figure 19: Network Static Stability for q(t) 53

Figure 20: Network Static Stability for $\Theta(t)$ 53

Figure 21: Network Dynamic Stability for u(t) 54

Figure 22: Network Dynamic Stability for $\alpha(t)$ 54

Figure 23: Network Dynamic Stability for q(t) 55

Figure 24: Network Dynamic Stability for $\Theta(t)$ 55
Figure 25: Input Characteristics 59
Figure 26: Plant Response 60
Figure 27: Prediction Error 60
Figure 28: Frequency Response for u(t) for Various Amounts of Training 61
Figure 29: Frequency Response for $\alpha(t)$ for Various Amounts of Training 61
Figure 30: Frequency Response for q(t) for Various Amounts of Training 62
Figure 31: Frequency Response for $\Theta(t)$ for Various Amounts of Training 62
Figure 32: Pseudo Random Binary Input Sequence and Spectral Content 63
Figure 33: Prediction Error for Fully Connected Network 65
Figure 34: Spectral Transfer Function for u(t)/Nonlinear Hidden Layer 68
Figure 35: Spectral Transfer Function for $\alpha(t)$/Nonlinear Hidden Layer 68
Figure 36: Spectral Transfer Function for q(t)/Nonlinear Hidden Layer 69
Figure 37: Spectral Transfer Function for $\Theta(t)$/Nonlinear Hidden Layer 69
Figure 38: Spectral Transfer Function for u(t)/Untrained Condition 71
Figure 39: Spectral Transfer Function for $\alpha(t)$/Untrained Condition 71
Figure 40: Spectral Transfer Function for q(t)/Untrained Condition 72
Figure 41: Spectral Transfer Function for $\Theta(t)$/Untrained Condition 72
Figure 42: Network Determined Control Input/Noise Rejection 75
Figure 43: Estimation and Tracking Error for u(t)/Noise Rejection 75
Figure 44: Estimation and Tracking Error for $\alpha(t)$/Noise Rejection 76
Figure 45: Estimation and TRacking Error for q(t)/Noise Rejection 76
Figure 46: Estimation and Tracking Error for $\Theta(t)$/Noise Rejection 77

ACKNOWLEDGEMENTS

A work such as this which involves the expenditure of large amounts of time and effort can never be accomplished through the labors of a single individual. As I sit down to complete this thesis, it is difficult to find the words to adequately express appreciation for the contributions of the many people who have made the completion of the task possible. Special thanks go to Professor Dan Collins, my thesis advisor, for providing me with the tools, both theoretical and physical, to accomplish this research. Thanks also to Professors Roberto Cristi and Jeff Burl for filling in the gaps in my knowledge of control theory, and to Professor Jim Hauser who served as the second reader for this thesis. Many thanks to the number of colleagues and friends with whom I shared my thoughts and from whom I received much guidance. Finally, a special thank you to my wife Eola and my daughter Amanda without whose patience and understanding none of this could have been possible.

I. INTRODUCTION

The classical control process involves eliciting a desired response from a known system. Determination of this known system is a non-trivial matter which can make the design of a control system difficult. As control algorithms become more powerful they require larger amount of a priori knowledge of system behavior. At the same time as systems become more complex the amount of uncertainty--plant variations, environmental disturbances, and random noise--which effects the system is increased. Two different approaches exist to handle this problem. The first involves designing robust controllers, controllers which provide good performance over a large range of uncertainty. The second, called adaptive control, involves a controller which alters the control law to compensate for system changes. The requirements of some current systems exceed the capabilities of either robust or adaptive control. The combination of these two approaches, robust adaptive control, is a promising new field of study.

Traditionally control of complex systems which require robustness and adaptation has been provided by human intervention. If human intelligence is the key to this type of control, perhaps a controller modelled on the capabilities of the brain may provide an alternative solution to the development of robust adaptive control. Neural networks are intended to provide a processing structure similar to the structure of the brain. The significant attributes of this structure are its parallel and distributed nature. This parallel distributed processing (PDP) structure is a natural form for the modelling of adaptive control problems. [Ref. 1]

This thesis will investigate the application of neural networks, also known as neurocomputing, to adaptive control. The purpose of this investigation is to develop and demonstrate a structure for the study of neural networks in adaptive control which is consistent with adaptive control theory. In Chapter II, neural networks are introduced and the Backpropagation neural network is presented. Chapter III describes the two separate

functions of adaptive control--estimation and control. From control theory a controller suitable for implementation in a neural network application is developed. The linear least squares estimation process is then introduced and the concept of a neural network as an estimator is described. Chapter IV combines the theories of neural networks and adaptive control to develop parallel distributed processing structures for estimation and control. Chapter V describes the setup of this computational experiment. The system of longitudinal motion for the A-4 aircraft is introduced as a system upon which to perform investigations in the use of neural networks for estimation and control. Chapter VI describes specific applications of these neural network structures to adaptive control. Chapter VII includes some concluding remarks and recommendations for further study. The objective of this thesis is to demonstrate the natural manner in which adaptive control problems can be represented using neural network structures.

II. NEURAL NETWORK THEORY

Neural networks represent a revolutionary new way of computing. Biological systems perform many tasks better than conventional computers. Artificial intelligence advocates believe that the development of powerful software is necessary to capture the power of the brain. Neural networks represent instead an attempt to imitate the capabilities of the brain in hardware. Neural networks are based on the idea that the brain utilizes a computational architecture different from that of the classical computer. An understanding of the basis for this architecture will provide a framework for investigations in neurocomputing. In this chapter, the theory of neural networks will be introduced. The concept of neural network processing is developed first on an intuitive level and then specifics of neural network architecture are presented. Following development of the neural network, the powerful Backpropagation neural network structure is introduced. With the understanding of these concepts, applications of neural networks to various problems may be easily understood. More thorough discussions of neural networks may be found in [Ref. 2] and [Ref. 3].

A. NEURAL NETWORK PROCESSING

Examining the operation of the brain in the context of computer processing may help to explain how neural networks work. The brain is a parallel processor. This parallelism allows the brain which operates at about 100 Hz to outperform computers operating at nearly 1,000,000 times that speed. Processing in the brain is parallel on a massive scale. The human neurological system contains billions of neurons. The brain is also a highly distributed processing system. Contrary to the conventional concept of a Central Processing Unit, the brain consists of numerous interrelated yet independent processors. These include not only the various regions of the brain but the millions of neurons which make up the senses. The processors in the brain are very simple compared to classical computers. By combining these traits--massive parallelism, distributed processing, and simplicity--neural networks hope to

emulate the problem solving characteristics of the brain. In this way, a neural network may be thought of as a Massively Parallel Distributed Processor. [Ref. 2:pp. ix-xi]

B. NEURAL NETWORK ARCHITECTURE

Every neural network paradigm is composed of the same architectural components. These structural and algorithmic factors include:

- A set of processing elements
- A state of activation for the network
- A pattern of connections
- An activation rule
- A propagation rule
- A learning rule
- An environmental interface

A network level illustration of the interaction of these elements is given in Figure 1. Computational processing elements which have some state of activation are connected in some pattern which interfaces with the environment through input and output elements. A depiction of the interactions of a single element is shown in Figure 2. Each element has an activation function and connections which are controlled by propagation and learning rules. These components will be described in detail below. [Ref.2:pp. 45-54]

1. Processing Elements

Neural networks are composed of a number of simple computational units called processing elements. All of these units act independently without the supervision of some global control. Simply put, a unit receives data from some of its neighbors on its input side, processes that data, then sends the result on to the same or other neighbors from its output side. The system is highly parallel because the operation of each element may occur simultaneously. Elements may have some physical significance or may be instead an abstract construct of the neural network. In this way, processing elements may be divided into three categories by their function. Input units represent access into the network from the physical

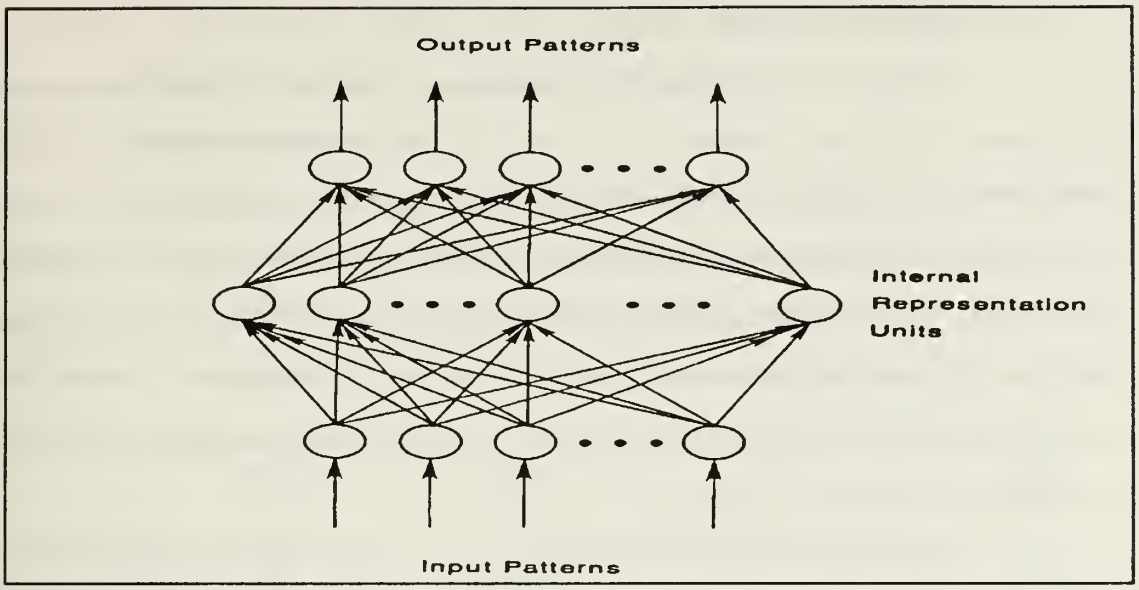

Figure 1: A Typical Neural Network Structure

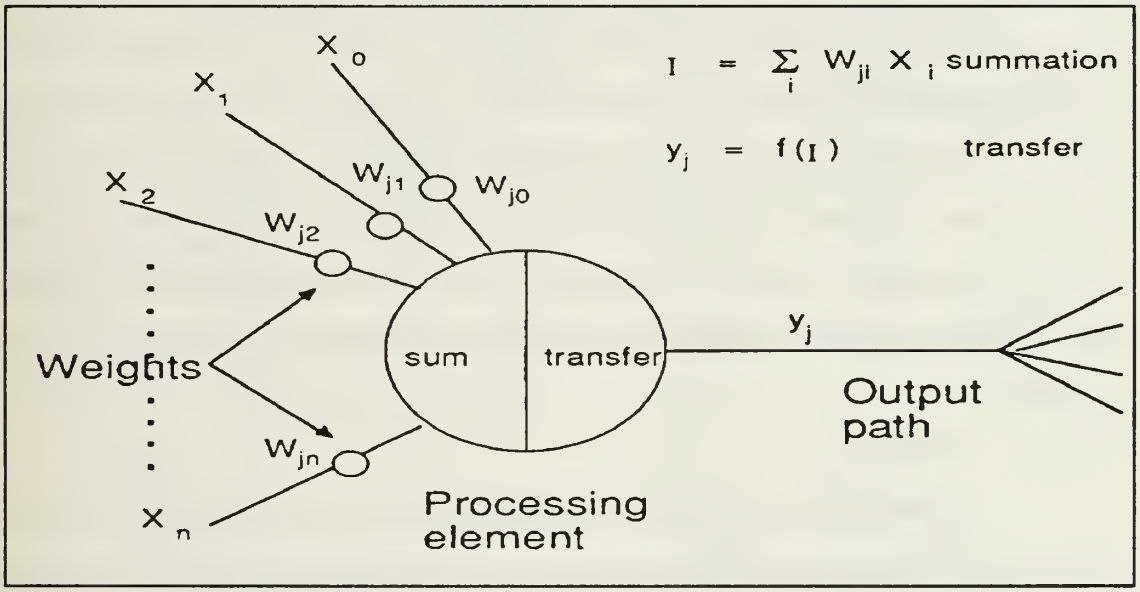

Figure 2: A Typical Processing Element

world. Output units represent the results of network operation. Hidden units are the abstract constructs developed by the system to solve the problem. From the viewpoint of control theory, input units may be considered to be elements of an input variable, output elements may be thought of as elements of an output variable, and hidden units may be conceived of as elements of a state variable. In this way, the neural network may be considered as a hardware implementation of state space and output equations. [Ref. 2:pp. 45-54]

2. State of Activation

The importance of these independent processors is that the elements represent subsystems of the total problem, which may be summarized by the global state of the network. At any moment in time, each processing element has a certain level of activation. It is the pattern and levels of these activations which represents the state of the system at a given time. The aggregate of these activations may be thought of as a multi-dimensional array which carves out some surface in space. Processing in this type of network may be thought of as evolving a system forward in time from some initial conditions to some steady-state value. [Ref. 2:pp. 45-54]

Considering possible interpretations of the number five is a simple example of the difference between this and classical computing. In a computer, the number five would be represented by some code stored in a location in memory. In a neural network, the number could be represented by the activations of five different processing elements, the total of which is five. The problem might actually be how to optimally store five tons of wheat in five grain elevators of different sizes. In this case, the activation of each element represents the amount of grain in that elevator. It should be apparent that this distributed processing form provides a more natural environment in which to solve certain classes of problems.

3. Connections

If the activations of the processing elements may be thought of as the states of the system, the connections may be thought of as the dependencies. These are the parts of the network which determine which processing elements react with each other and in what manner. A connection may be envisioned as taking the activation from the output side of a processing element, operating on it, and transporting it to the input side of a neighboring element. In most cases, the contribution of each element is considered to be additive. Therefore, the input of any unit may be considered to be the weighted sum of the activations of the units connected to its input side. More complex input weighting functions involving

products as well as summations have also been proposed. The structural dependencies of the network are determined by these connections. [Ref. 2:pp. 45-54]

4. Activation Rule

The activation rule determines the activation value of the processing element given a set of inputs. Figure 3 gives an example of the complexity of the activation function from the Neuralworks Professional II neural network development software. Activations may be discrete or continuous, bounded or unbounded, stochastic or deterministic. Activations

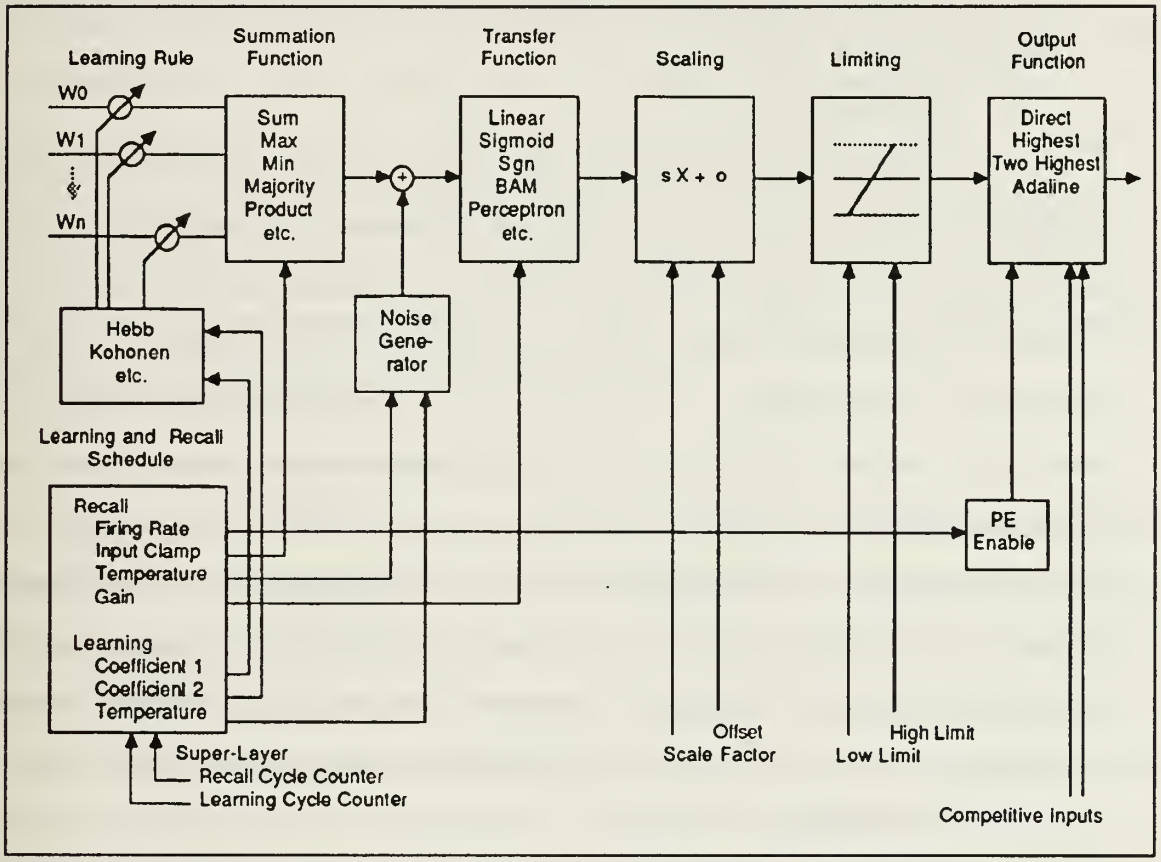

Figure 3: Neuralworks Professional II Activation Function Logic

may contain a summation function, a transfer function, scaling, limiting, thresholding, and competition. Through the use of complex activation functions the network may model a variety of nonlinear systems. [Ref. 3:pp. 146]

5. Propagation Rule

The propagation rule is the precept which dictates how the activations of units are transferred to the inputs of other units. All neural networks have a propagation rule. This may be thought of the procedure for feed forward operation of the network. This rule ties input elements, connections, and output elements together. It governs not only the connection function, but the order in which connections are made, both architecturally and by type. This succession may be sequential, by layer, random, or perhaps by the properties of the connections or elements. The propagation rule regulates the feed forward operation of the network. [Ref. 2:pp. 45-54]

6. Learning Rule

As the propagation rule may be thought of as the rule for the feed forward behavior of the network, the learning rule may be thought of as the feedback rule. Some networks do not use feedback and therefore require no learning rule. For networks which do learn, this complex function changes the structure of the network to reflect given interactions at each instant. The learning rule may change either the activations of the networks processing elements or the connections between elements or both. This change is based on the state of activations of the network, the network connection weights, and often some desired result. Networks which attempt to map some input into a desired result are known as supervised learning networks. Usually, the activations of the elements are allowed to change during feed forward operation governed by the propagation rule while the learning rule changes the connection weights during feedback operation. [Ref. 2:pp. 45-54]]

7. Environment

The interface between the neural network and the environment is as important as the network itself. Determination of such things as the number of inputs, the timing of inputs, and the input character itself is extremely important. A neural network is not a panacea for all problems, but a new form of processing tool to be used in applications where both the problem and parallel distributed processing are understood. The consequence of the

importance of these design considerations is that the user of a neural network must be familiar with both neural networks and the domain of the problem to be solved. [Ref. 2:pp. 45-54]

C. NEURAL NETWORK OPERATION

A neural network consists of numerous processing elements which are connected to the environment through input and output units. These elements are attached by weighted connections which are meant to represent some form of dependency. Each element also has an activation function. Network operation consists of a feed forward phase and in some cases a feedback phase. The feed forward phase is governed by the propagation rule. In this phase, the activation of each element is altered in response to its connections, activation function, and some input. The learning rule controls the feedback phase. In this phase, the connection weights of an element are changed in response to its connections, the network's state of activation, and some desired result. Once this is complete, the process is repeated until the desired outcome is achieved. [Ref. 3:pp. 3-10]

D. BACKPROPAGATION ALGORITHM

In the 1950's and 1960's, neurocomputing was in its infancy. Many successful, though limited, applications of neural networks were developed. Most of these involved networks using simple activation rules grouped into input and output layers. In 1969 Minsky and Papert, two MIT artificial intelligence researchers, published a book called Perceptrons in which they showed that networks must use hidden layers in order to sufficiently solve most problems. Unfortunately, at the time, no learning rule or activation rule capable of exploiting the power of hidden layers existed. Because of this, neurocomputing went into a state of dormancy for nearly thirty years. One major reason for the resurgence of interest in neural networks is the development of the Backpropagation paradigm, a type of network which successfully uses hidden units. The architecture, learning rule and activation functions of a Backpropagation neural network combined give this network the capability to utilize hidden layers in order to solve complex problems. [Ref. 2:pp. 111-112]

1. Architecture

A Backpropagation neural network is laid out in a relatively simple, straightforward manner. The processing elements are arranged in a number of parallel layers including an input layer, an output layer, and any number of hidden layers. The elements in each layer are usually fully interconnected with the elements in a previous layer as in Figure 1. Any N-dimensional feature space may be represented with the use of a suitable activation function and a sufficient number of hidden units [Ref. 4]. Operation of the network involves presentation of a set of input-output pairs or patterns. This process is known as supervised learning. The network first feeds forward the input to the hidden and output layers. Then in the feedback phase, the error between the network produced result and the desired output is used to alter the connection weights, or dependencies of the network. This process may be imagined alternatively as carving out some N-dimensional feature space, performing some massively parallel regression, or encoding the inputs into state variables which are in turn combined to give outputs. It is the ability of the Backpropagation neural network to capitalize on the capabilities of hidden units to represent any feature space which makes it so powerful. [Ref. 2:pp. 318-328]

2. Backpropagation Learning Rule

The power of hidden layers is useless without a learning rule which can utilize them. One problem with hidden layers has been determining how to distribute responsibilities for network error. The Backpropagation learning rule does this by minimizing a global cost function with respect to the connection weights in a least squares sense. This is an implementation of a gradient descent procedure on the error surface in weight space. The application of this concept to output layers is straight forward, however, extension of the concept to hidden layers is more complex. [Ref. 2:pp. 318-328]

For elements in the output layer, minimization of the square of the errors is derived in the traditional least squares fashion. The global square error cost function, J, may

be expressed as

$$J = \tfrac{1}{2} \sum_k \sum_o (T_o(k) - X_o(k))^2 \tag{2.1}$$

where $T_o(k)$ denotes some target output for pattern k and $X_o(k)$ indicates the activation of output element o resulting from pattern k. The output (activation) of an element is produced using a function of the net input to the element

$$X_o(k) = f(I_o(k)) \tag{2.2}$$

where $I_o(k)$ represents the net input to element o given by

$$I_o = \sum_i (w_{oi} X_i) \tag{2.3}$$

where w_{oi} is the weight connecting element i to element o and X_i is the activation of element i. The function f is known as the activation function. By changing the weights in proportion to the derivative of the cost function, the error may be minimized. If the cost function is to be minimized for each instant (pattern) and with respect to individual weights, the derivative of J may be taken inside of the summations, allowing the subscripts to be dropped. The derivative of the cost function can be broken down into three elements using the chain rule

$$\frac{\partial J}{\partial w_{oj}} = \frac{\partial J}{\partial X_o} \frac{\partial X_o}{\partial I_o} \frac{\partial I_o}{\partial w_{oj}} \tag{2.4}$$

where the first term denotes the derivative of the cost function with respect to the activation function, the second term denotes the derivative of the activation function with respect to the net input to the element, and third term denotes the derivative of the net input to an element with respect to the weights entering the element. Solving for these terms gives

$$\frac{\partial J}{\partial X_o} = (T_o - X_o) \tag{2.5}$$

the error between the target output and the element activation for the first term using equation (2.1)

$$\frac{\partial X_o}{\partial I_o} = f'(I_o) \tag{2.6}$$

the derivative of the activation function for the second term using (2.2), and for the third

term using (2.3)

$$\frac{\partial I_o}{\partial w_{oj}} = X_o \tag{2.7}$$

the activation of the element. By combining (2.5), (2.6), and (2.7) and adding a constant of proportionality, α, the basis of the Backpropagation learning rule is defined

$$\Delta w_{oj} = \alpha \, \delta_o \, X_o \tag{2.8}$$

where δ_o is an error term defined as the combination of (2.5) and (2.6)

$$\delta_o = f'(I_o) \, (T_o - X_o) \tag{2.9}$$

The constant of proportionality, α, is known as the learning rate. For elements in the output layer, this algorithm is relatively straight forward. But how can δ be defined for hidden layers? This is the error assignment problem the solution of which makes Backpropagation such a powerful technique. [Ref. 2:pp. 318-328]

The Backpropagation network assumes that each processing element is in some way responsible for the error in the output. The learning rule operates much as the name implies to distribute the error. By "backpropagating" the error along the same connections and with the same weights used in the feed forward cycle, the network assigns a portion of the error to each processing element. The learning rule for hidden layers is identical to that for the output layer (2.9), with the exception that δ for hidden units is defined as the derivative of the activation function multiplied by the δ backpropagated from the previous layer, or

$$\delta_j = f'(I_j) \sum_o (w_{oj} \, \delta_o) \tag{2.10}$$

where the summation is over the elements to which the hidden element j is connected. The subscript o need not denote the output layer, but can denote any layer to which the output of element j is connected. The Backpropagation network can thus be said to operate in a similar manner to other neural networks. First, the input is fed forward through the network to determine the output activations. These activations are then compared to the desired outputs and the error is fed back through the network. Finally, the weights connecting the

elements are changed using the learning rule and the fed back error. The true power of the Backpropagation network lies in its ability to use hidden layers. [Ref. 2:pp. 322-328]

One drawback to the use of a gradient descent method for learning is that the network training follows a very jagged path in the error-weight space. This is obvious when one considers that the probability of two consecutive error vectors in weight space pointing in the same direction is zero. Therefore, the network tends to wander about in weight space, oscillating across the optimal path to a global minimum. One solution to this problem is to average the current estimate of the proper direction with past estimates. This is accomplished by adding what is known as a momentum term to (2.9)

$$\Delta w_{oj}(t+1) = \alpha \, \delta_o \, X_o + \mu \, \Delta w_{oj}(t) \tag{2.11}$$

where μ is another constant of proportionality. In the past α and μ have been determined empirically, however the results of recent research recommend a number of ways to statically and dynamically determine appropriate values for these constants. [Ref. 2:pp. 329-330]

3. The Backpropagation Activation Rule

The activation function used in this type of neural network must be compatible with the learning rule described above. This implies that the activation rule must be differentiable over the entire range of possible values. Another desirable characteristic of an activation function is that the function have a unique output value for a given input value. Monotonic functions have unique output values for given inputs with the added advantage that the behavior of the activation function is predictable (an increase in the input always implies an increase in the output.) Typically, the input to a processing element (2.3) is defined as the weighted sum of the activations of the elements connected to its input side

$$I_i = G * \Sigma_j (w_{ij} X_j) \tag{2.12}$$

Where G is some gain added as a scaling factor. At first glance, a linear activation function appears to be ideal. The output value, or activation, of a processing element using the linear

activation function is simply

$$f(I) = I \tag{2.13}$$

where the subscript i is dropped for convenience. The derivative of this function is simply a constant, which can be absorbed in the learning rate (proportionality constant) defined above. However, as Minsky and Papert proved, linear networks are only capable of representing linear systems. They can not exploit the power of hidden layers. Neural networks utilizing the semi-linear functions capitalize on the strengths of the linear function and the use of hidden layers. Two semi-linear functions are the sigmoid and hyperbolic tangent functions. For the sigmoid function (shown in Figure 4), the activation is

$$f(I) = \frac{1}{1 + e^{-I}} \tag{2.13}$$

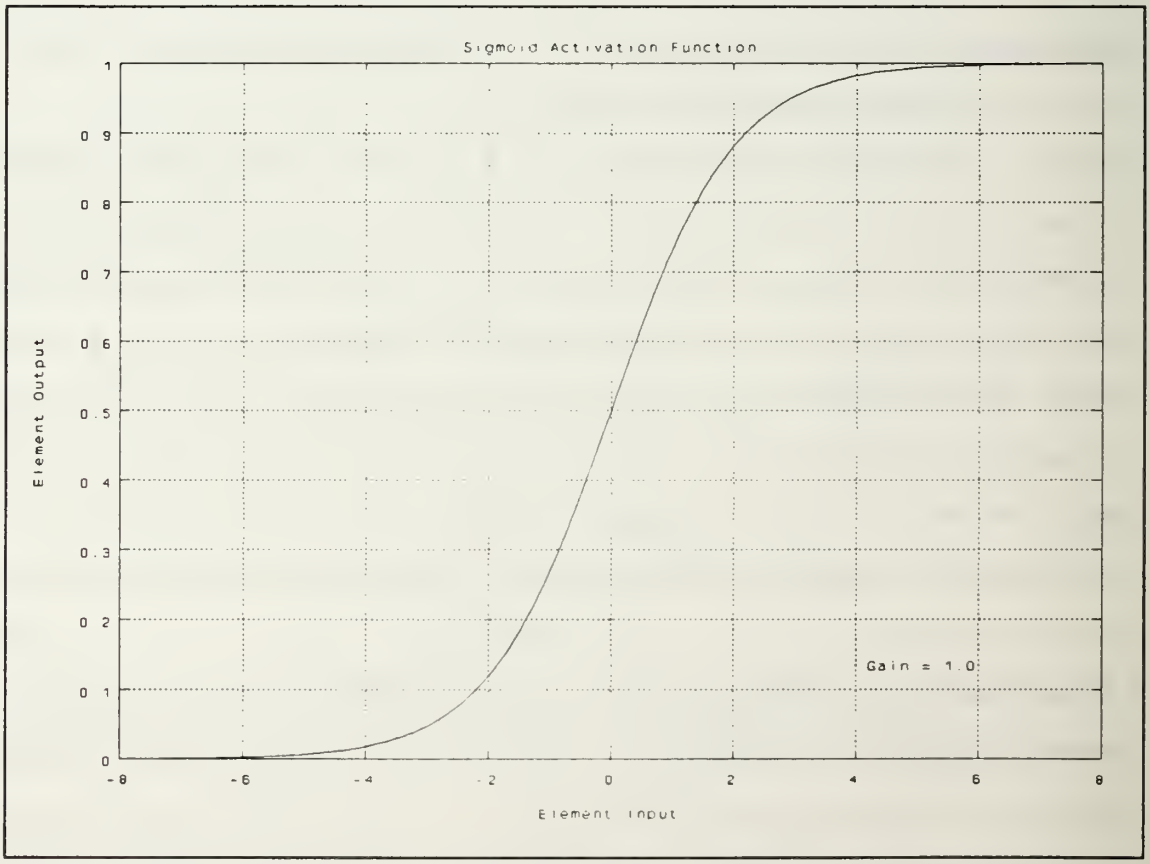

Figure 4: Plot of the Sigmoid Activation Function

and the derivative is

$$f'(I) = \frac{e^{-I}}{(1 + e^{-I})^2} \tag{2.14}$$

or in terms of the original function

$$f'(I) = f(I) * (1 - f(I)) \tag{2.15}$$

Notice the sigmoid is limited to values between 0 and 1. Perhaps a better activation, due to its range, is the hyperbolic tangent activation function (shown in Figure 5)

$$f(I) = \frac{e^{I} - e^{-I}}{e^{I} + e^{-I}} \tag{2.16}$$

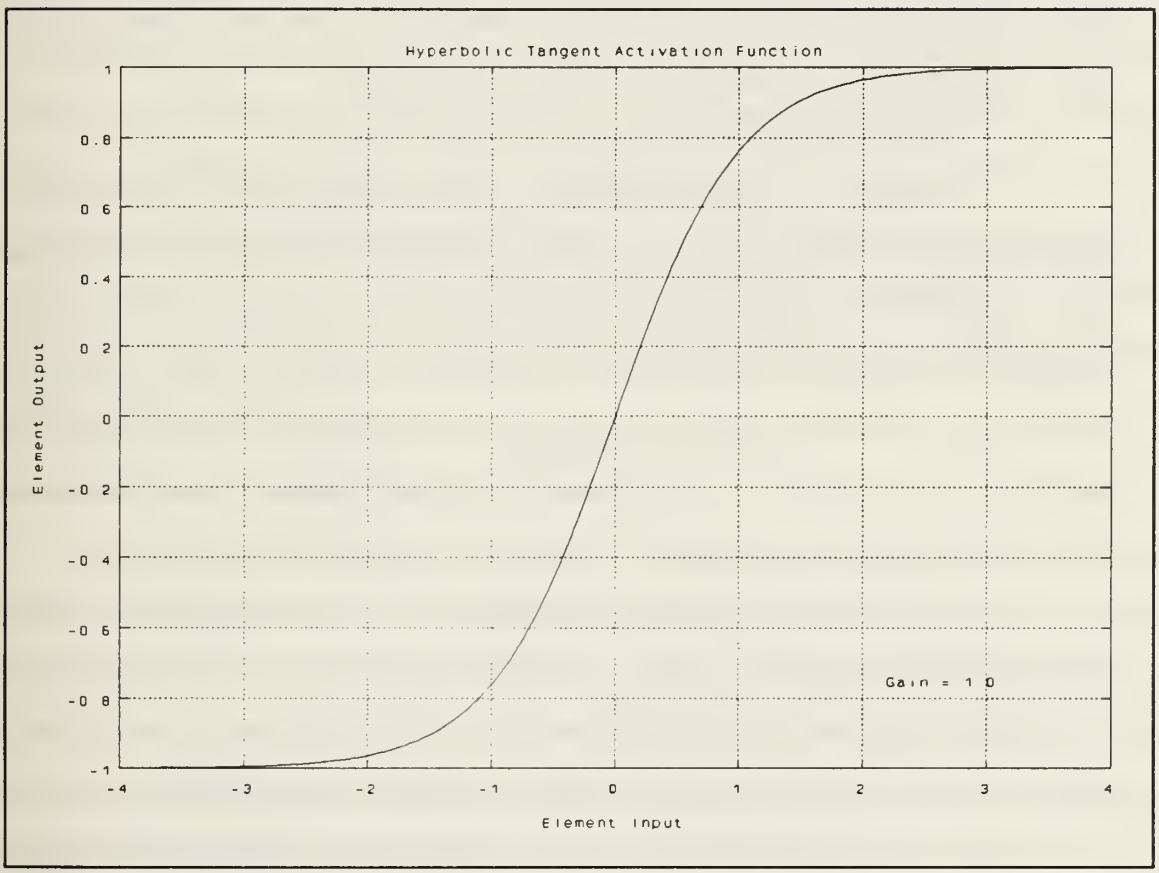

Figure 5: Plot of the Hyperbolic Tangent Activation Function

whose derivative expressed in terms of the original function is

$$f'(I) = (1 + f(I)) * (1 - f(I)) \tag{2.17}$$

These activation functions share two very important qualities. First, by expressing the derivative in terms of the activation function, a large amount of computation time is saved. Second, these non-linear functions approximate linear functions over some central region while guaranteeing stability at its extremities. These are the most widely used activation functions. [Ref. 3:pp. 161-163]

Another proposed activation function is the sine wave. This may be thought of as performing some type of "Generalized Fourier Analysis." It is thought that a neural network using sine wave activation functions may perform some modal decomposition, discovering important spectral components of the function described by the input-output pairs. Research is ongoing in this area. [Ref. 3:pp. 449-450]

4. The Power of the Backpropagation Neural Network

The power of the Backpropagation neural network rests in its extension of classical methods to the use of hidden constructs. By using the concept of a gradient descent search algorithm, the Backpropagation neural network incorporates all of the theory developed for these types of algorithms. At the same time, using parallel processing and hidden layers, the Backpropagation neural network is capable of extending its scope beyond that of linear systems to that of any N-dimensional system. The are a number of similarities between the Backpropagation network and traditional methods of estimation.

In this chapter, the theory of neural networks was introduced to provide an understanding of this processing tool. An intuitive approach was first described, followed by a detailed description of the building blocks of a neural network. Following a summary of the operation of a neural network, the Backpropagation neural network was introduced and important characteristics described. Finally, the power of the Backpropagation neural network was outlined. As will be seen in the next chapter, the concepts behind Backpropagation are very closely related to those of adaptive control, especially the estimation process.

III. ADAPTIVE CONTROL THEORY

From a general perspective, an adaptive controller is one which changes the control as it perceives changes in the environment or system. The need for adaptive techniques arises when a system and its environment can not physically or practically be completely specified. The general definition given above suggests that adaptive control may be divided into two functions--a model estimation function and a control function. The basic layout of an adaptive controller is given in Figure 6. The separability of the two tasks illustrated in the figure permits the adaptive control designer to select from numerous control techniques and estimation methods. For the purposes of this thesis, the concentration will be on the control and estimation of deterministic systems. These are systems in which noise is relatively unimportant with respect to modelling errors. In this chapter the relatively simple one step ahead control algorithm will be used to introduce a general controller followed by an examination of the linear least squares estimator. Further information on the topic of adaptive control is available in [Ref. 5] while specific information on estimation is available in [Ref. 6]. As a summary, the natural way in which neural networks represent these techniques will be delineated.

A. ONE STEP AHEAD PREDICTION CONTROL

One step ahead prediction control is defined as that control necessary to bring a system to some desired state in one step. For most control applications it is assumed that the system under consideration is linear, time-invariant, and causal. Of the many equivalent models of this type for a system, the simplest to use in developing adaptive control algorithms is the discrete time deterministic autoregressive moving average (DARMA) model. The DARMA model is characterized by the equation

$$A(q)\,\underline{y}(t) = B(q)\,\underline{u}(t) \tag{3.1}$$

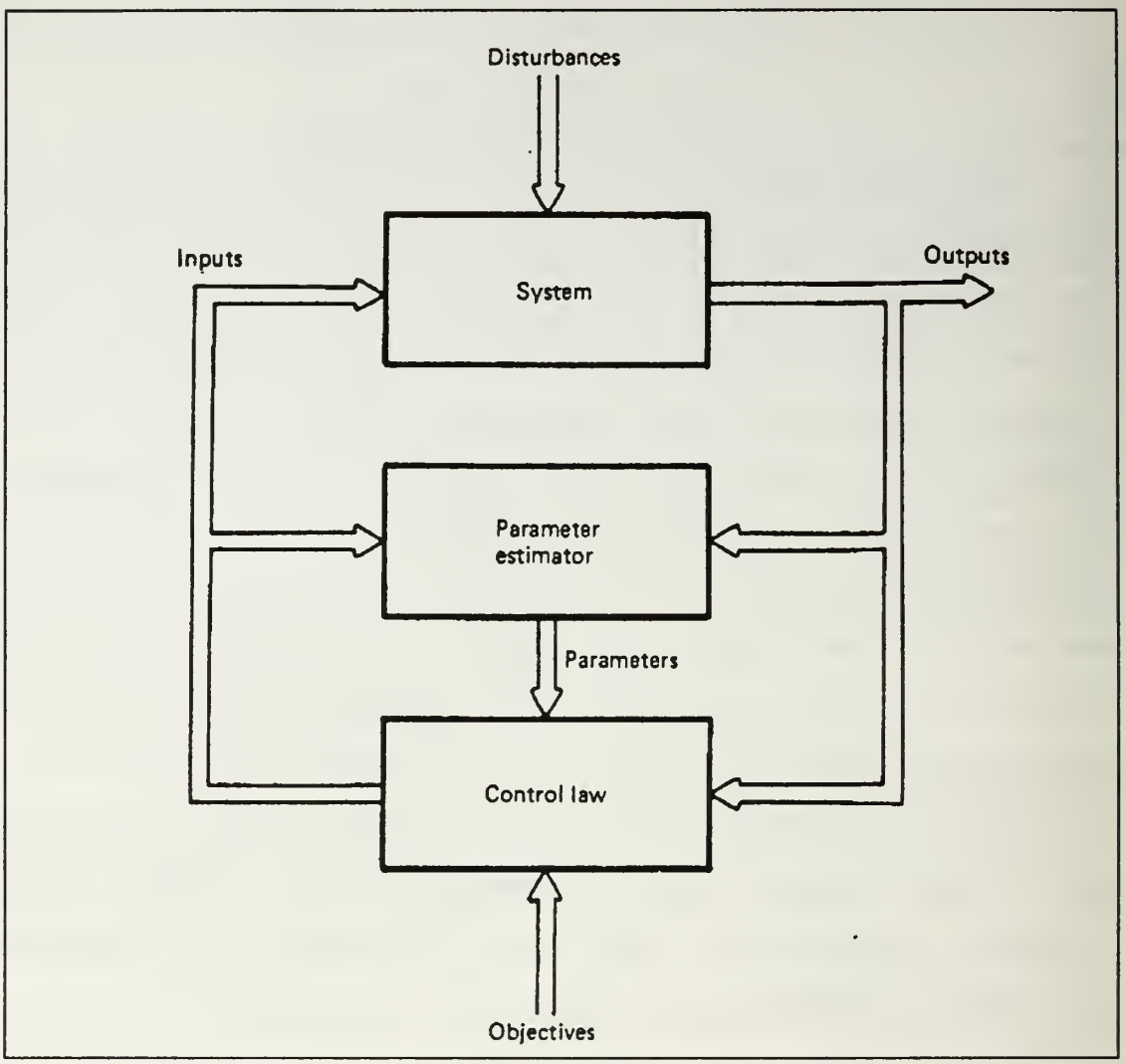

Figure 6: Adaptive Control Structure

where

$$A(q) = I + A_1(q) + A_2(q) \ldots$$

$$B(q) = B_0 + B_1(q) \ldots$$

and A(q) and B(q) are matrix polynomials in the backward shift operator, q^{-1}, $\underline{y}(t)$ is the system output, and $\underline{u}(t)$ is the system input. The DARMA model is roughly equivalent to a transfer function or a controllable-observable state space representation of the system [Ref. 5:pp. 7-40]. If a single input single output (SISO) DARMA model is expanded in the shift

operator and rearranged the equation becomes

$$y(t) = b_1u(t-1) + b_2u(t-2) \ldots - a_1y(t-1) - a_2y(t-2) \ldots \quad (3.2)$$

This equation can be used as a predictor for the output at the next time step

$$\hat{y}(t+1) = b_1\, u(t) + b_2\, u(t-1) \ldots - a_1\, y(t) - a_2\, y(t-1) \ldots \quad (3.3)$$

where $\hat{y}(t+1)$ indicates the predicted value for $y(t+1)$. Equation (3.3) can be used to determine the control input required to bring the system to a desired value $y_d(t+1)$ in one step by replacing the variable $\hat{y}(t+1)$ with the desired value $y_d(t+1)$ and solving the equation for $u(t)$

$$u(t) = 1/b_1 \{y_d(t+1) + a_1\, y(t) + a_2\, y(t-1) \ldots - b_2\, u(t-1) \ldots \} \quad (3.4)$$

This is known as one step ahead prediction control. [Ref. 5:pp. 118-171]

The one step ahead prediction control law is the result of the minimization of the quadratic cost function

$$J(t) = \tfrac{1}{2} \{ y(t+1) - y_d(t+1) \}^2 \quad (3.5)$$

A variety of cost functions of the same form may be developed using different forms of the input and output variables. The consequence of this is that the control law in (3.4) may represent any number of different control strategies. [Ref. 6:pp. 461-481]

Using this concept of many strategies being represented by one form of controller introduces the idea of a general control structure. In (3.4) the $y_d(t+1)$ term may be conceptualized in a number of different ways. In general, it may be thought of as some reference input to the system. This implies some type of tracking control. If this reference input is generated by some reference model, the one step ahead controller becomes a model reference (MR) controller. If the past values of $u(t)$ and $y(t)$ are thought of as state variables, the one step ahead controller becomes some type of state variable feedback with a reference input controller

$$u(t) = K(t)\underline{x}(t) + r(t) \quad (3.6)$$

All of these controller types have the same basic structure, the only difference is in the determination of the relevant parameters. The vector of past inputs and outputs in equation

(3.4) becomes the state variable in equation (3.6) to provide a controller for an adaptive algorithm. [Ref. 5:pp. 118-171]

One step ahead control provides a simple method of introducing a general controller structure. Using the vector of past input and output measurements as some state vector, the idea of a controller based on the weighted sum of state variables and a reference input may be developed. Equation (2.3) defines the input to a neural network processing element to be the weighted sum of the element activations connected to its input side. Thus a neural network processing element may in some way represent this general form of controller.

B. LINEAR LEAST SQUARES ESTIMATION

Adaptive control is composed of two functions, estimation and control. Numerous techniques exist to develop estimators for systems, however, the majority involve extensive off line computation. By far the largest category of on line estimation techniques develops estimates of the system parameters based on minimization of quadratic cost functions similar to that used in the development of the one step ahead controller. By deriving one form of these linear least squares estimators, the recursive least squares method, a general structure for estimation will be developed.

The derivation commences with the SISO version of the DARMA model introduced in equation (3.1). Upon expansion, this expression is equivalent to

$$y(t) = \sum_j b_j\, u(t-j) - \sum_k a_k\, y(t-k) \tag{3.7}$$

where the index j represents dependencies on past input measurements and the index k represents dependencies on past output measurements. Since this is a linear, time invariant system the coefficients, a_k and b_j, and the past input-output data may be grouped separately to give

$$y(t) = \theta^T \phi(t-1) \tag{3.8}$$

where θ is called the parameter vector defined by

$$\theta = [\, b_0\ b_1\ b_2 \ldots a_1\ a_2\ a_3 \ldots] \tag{3.9}$$

and $\phi(t-1)$ is a the regression vector containing past measurements of the input and output

$$\phi(t-1) = [u(t)\ u(t-1)\ u(t-2)\ \ldots\ -y(t-1)\ -y(t-2)\ -y(t-3)\ldots] \tag{3.10}$$

Using this relationship, the value of y(t) at a future time may be predicted as in (3.3)

$$\hat{y}(t) = \theta^T \phi(t-1) \tag{3.11}$$

The error in this prediction is used in a quadratic cost function to determine some optimal value for θ. [Ref. 6:pp. 51-59]

A linear, time-invariant, deterministic system has a single parameter vector but many regression vectors, each representing measurements made at a given time step. Therefore, a quadratic cost function in the prediction error can be formed using equation (3.11) and the measured output at a given time t

$$J(\theta) = \tfrac{1}{2} \sum_t (y(t) - \theta^T \phi(t-1))^2 \tag{3.12}$$

where t covers the range of the N measurements made. Differentiating with respect to the parameter vector, θ, and setting the result equal to zero results in an expression of the form

$$\Theta = [1/N \sum_t \phi(t)\ \phi^T(t)]^{-1}\ 1/N \sum_t \phi(t)\ y(t) \tag{3.13}$$

where Θ is the estimated parameter vector [Ref. 6:pp. 176-181]. Making some assumptions on the content of the result of this summation and applying the matrix inversion lemma three equations for determination of the parameter vector Θ are obtained

$$\Theta(t+1) = \Theta(t) + L(t)\ (y(t) - \Theta^T(t)\ \phi(t-1)) \tag{3.14}$$

$$L(t) = P(t-1)\ \phi(t-1)\ [1 + \phi^T(t-1)\ P(t-1)\ \phi(t-1)]^{-1}$$

$$P(t) = P(t-1) - \frac{P(t-1)\ \phi^T(t-1)\ \phi(t-1)\ P(t-1)}{1 + \phi^T(t-1)\ P(t-1)\ \phi(t-1)}$$

where P at time zero is some positive, definite matrix representing the confidence in an initial estimate Θ at time zero. The first equation is a predictor-corrector equation while the second and third equations represent some method of changing the estimation gain. This is known as the recursive least squares estimation method. The equations (3.14) are easily expanded to the multivariable case. This is a very robust form of estimator. [Ref. 6:pp. 303-310]

A more general form of this predictor corrector equation is the basis for all least squares parameter estimation schemes. This equation is

$$\Theta(t+1) = \Theta(t) + M(t)\,\phi(t-1)\,e(t) \tag{3.15}$$

where the terms in the equation are defined as

M(t) - the algorithm gain

$\phi(t-1)$ - the regression vector

e(t) - the prediction error

Similar in context to the general structure developed above for controls, the gain, regression vector, and prediction error used in the least squares estimator result from the particular cost function which is minimized and the various assumptions made about the character of the estimation process. The gain term can vary from a scalar constant to a large covariance matrix as seen in (3.14). The regression vector and prediction error may likewise take on a number of different forms. [Ref. 5:pp. 47-100]

One significant factor in this derivation is the fact that the Backpropagation learning rule in equation (2.8) is similar to the general form of the linear least squares parameter estimator in equation (3.15). In fact, a neural network with a linear activation function is a parallel distributed processing implementation of the general linear least squares estimator. The theorems and proofs applicable to least squares estimation are in a general sense applicable to the Backpropagation neural network. Another major factor concerns the structured form of both the estimation algorithm developed above. This suggests that the capability of neural networks to naturally represent structured problems may prove useful in estimation applications.

C. NEURAL NETWORKS AND ADAPTIVE CONTROL

By combining the control and estimation algorithms described above, an adaptive controller may be constructed. The control law determines control inputs using the estimated model as if it were the true model. More importantly from the point of view of neural network applications is the general structure developed for control and estimation. For the

controller, the input is generated as some weighted sum of terms in a regression vector. In the case of model reference adaptive control (MRAC) shown in Figure 7 these weights are determined by minimizing the error between the network predicted output and the model reference output. [Ref. 5:pp. 199-202] For the estimator, the predicted output is also some weighted sum of the terms in a regression vector. In this case, the weights are altered to minimize the error between measured and predicted output. Considering these factors as inputs, weights, and outputs it appears that a Backpropagation neural network would provide an intuitive structure for the solution of adaptive control problems.

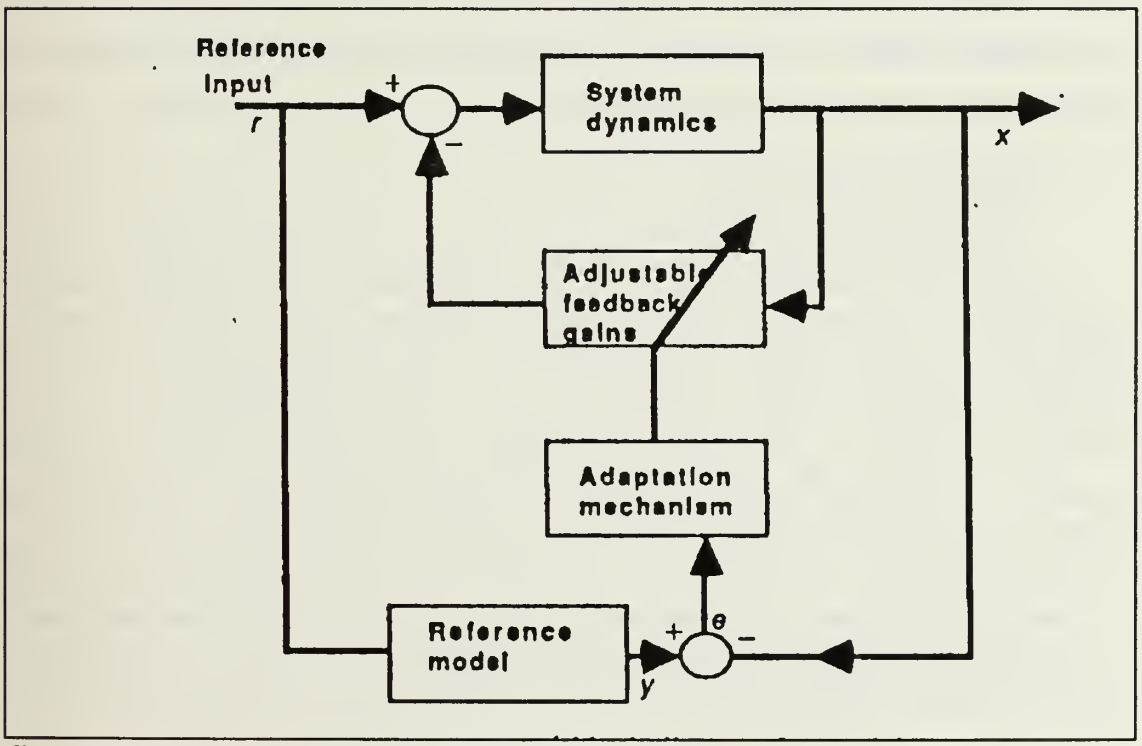

Figure 7: Model Reference Adaptive Control Structure

IV. NEURAL NETWORK ADAPTIVE CONTROL STRUCTURES

In this chapter the concepts of Backpropagation and adaptive control developed thus far will be combined to produce two structures for neural network adaptive control. As earlier introduced, adaptive control can be divided into two separable tasks, estimation and control law synthesis. The estimation process involves the mapping of inputs to outputs. The ability of a Backpropagation neural network to accomplish this is evident. The application of Backpropagation to control law synthesis is more complex. The idea of a generic control law involving the weighted sum of state variables and a reference input was advanced in the discussion of control theory. At the same time, the notion of Model Reference Adaptive Control was presented to provide some target output. Linking these concepts, the control law synthesis may be viewed as the mapping of state variables and reference inputs to a model reference output through some predetermined model. In the paragraphs that follow, two methods of separating the tasks of estimation and control in a Backpropagation neural network structure are proposed. The first involves a spatial separation, the functions are performed in parallel. The second utilizes temporal separation with the estimation and control duties accomplished sequentially.

A. PARALLEL STRUCTURE

In the parallel approach to neural network adaptive control, the estimation and control responsibilities are physically separated within the neural network. This is similar to the principle of certainty equivalence [Ref. 7]. A network demonstrating this structure is presented in Figure 8. The network on the left in the figure performs the estimation function. The purpose of this network is to map measured inputs to measured outputs. Though this network is composed of a single linear layer, multiple layers with varying connections and activation functions could be used. As long as the composition of the input and output layers remains constant, the internal structure can be varied in any manner.

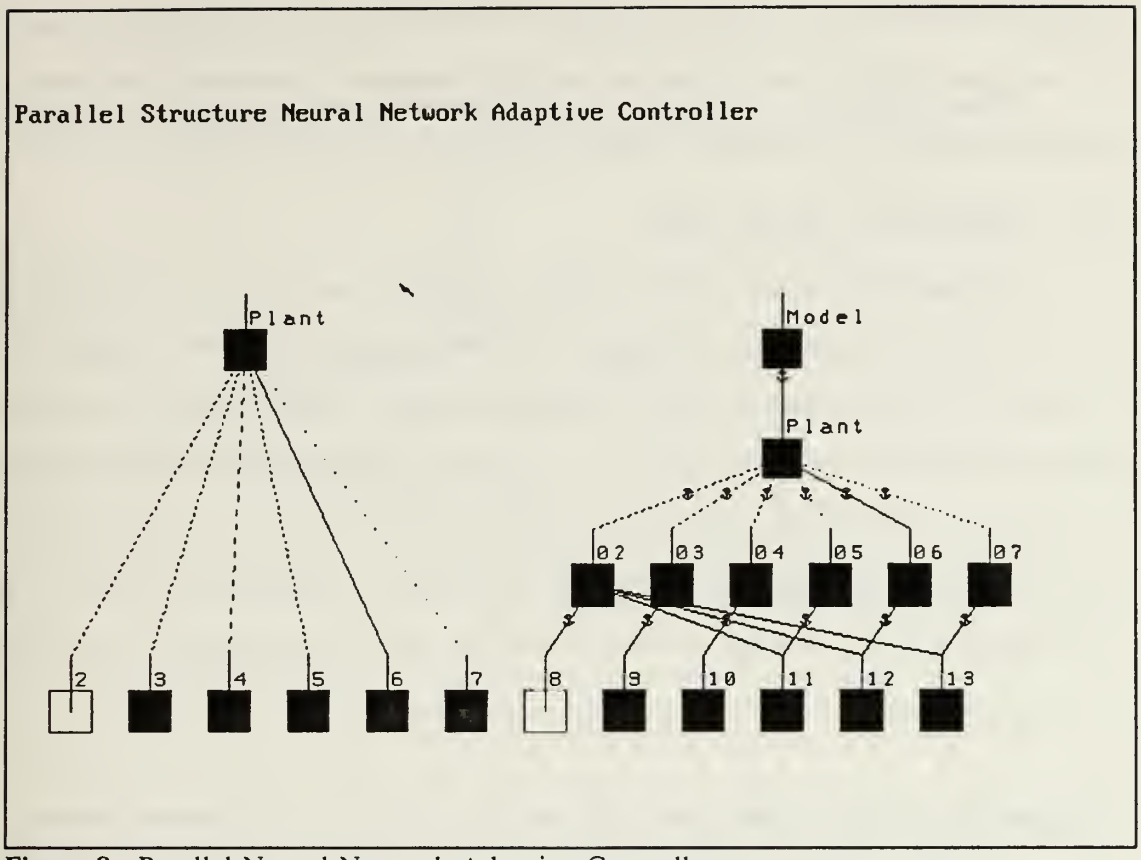

Figure 8: Parallel Neural Network Adaptive Controller

Control law synthesis is provided by the network on the right in Figure 8. This network contains two structures. The middle two layers are an exact duplicate of the estimation network on the left, and represent a black box model of system operation. The weights in this structure are fixed as far as the control process is concerned. They are obtained through links to the estimation network. As the estimation process progresses, the model in the control network is continually updated. The control process is represented by the two external layers added to this model. The input layer is composed of the state variables and reference input whose weighted sum makes up the control input. The additional output layer provides the desired model reference output. This network maps the state variables and reference input into a control input which is propagated through the internal model. The resultant predicted output is compared to the reference output and the error is backpropagated through the model. This error is then used to change the weights,

or gains, on the controller inputs. In essence, the estimation network generates a simulation of the system while the control network used this simulation to generate control inputs. The parallel nature allows the network designer a large amount of flexibility.

B. SEQUENTIAL STRUCTURE

The sequential neural network adaptive controller structure is a derivative of the control network on the right in Figure 8. Since the structure for the estimator and the controller exist in the control network, both functions can be performed by one network if they were separated temporally [Ref. 8]. The basic concept for a linear network applied to a third order SISO system is demonstrated in Figure 9 and Figure 10. Operation of this network involves completion of the estimation function as a measurement is made followed by generation of the next control input during the inter sample period. In the estimation

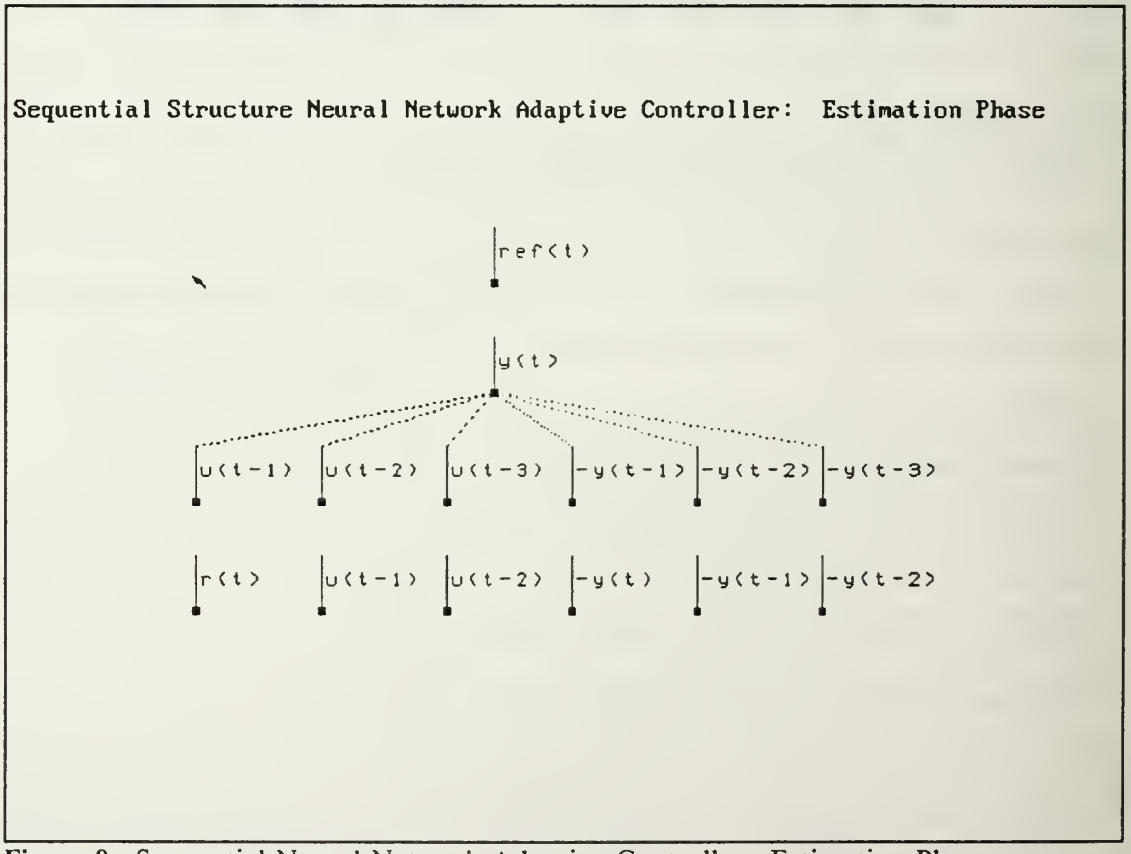

Figure 9: Sequential Neural Network Adaptive Controller: Estimation Phase

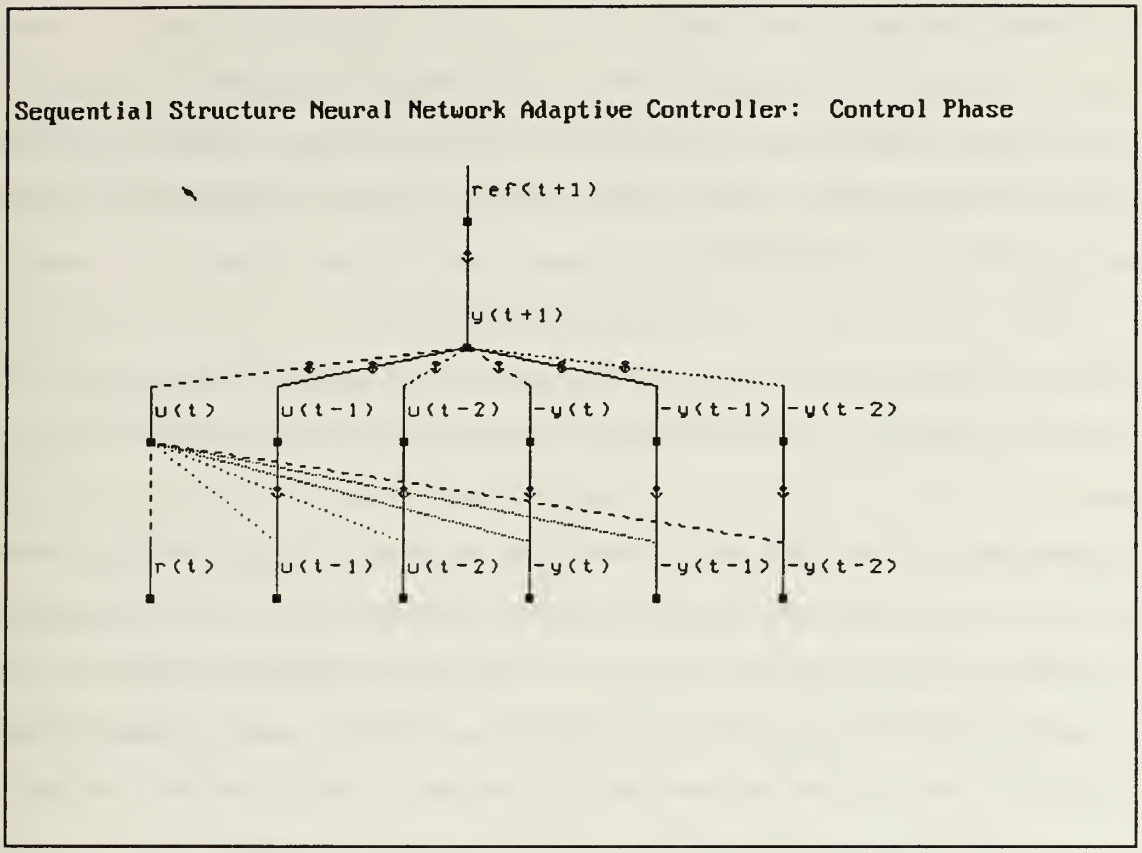

Figure 10: Sequential Neural Network Adaptive Controller: Control Phase

phase, the outer layers of the network in Figure 9 are inactive. At a given sample time, a measurement of the output of the system, y(t), is made. It is assumed that the control inputs, u(t), are known. Using three past measurements of the output and three past known control inputs, an input vector for the network is assembled similar to the regression vectors discussed in Chapter III

$$\phi(t-1) = [u(t-1)\ u(t-2)\ u(t-3)\ -y(t-1)\ -y(t-2)\ -y(t-3)] \qquad (4.1)$$

This is the vector which is applied to the lower middle layer of the network in Figure 9. This input vector is propagated forward through the weights in the network. Since the activation function of the element labelled y(t) is linear, the activation of the element is equal

to its input or

$$\hat{y}(t) = w_{21}\, u(t-1) + w_{22}\, u(t-2) + w_{23}\, u(t-3) - w_{24}\, y(t-1) - w_{25}\, y(t-2) - w_{26}\, y(t-3) \quad (4.2)$$

where w_{ij} is the weight connecting element j in layer i to the output element y(t) and $\hat{y}(t)$ is the network prediction for y(t). These weights are identical to the system coefficients contained in the parameter vector

$$\Theta(t) = [\, b_1\ b_2\ b_3\ a_1\ a_2\ a_3\,] \quad (4.3)$$

The error between the measured value for y(t), used as a target for the network, and the predicted value $\hat{y}(t)$ is backpropagated through the network using the learning rule

$$\Delta w_i = \alpha\, X_i\, \delta \quad (4.3)$$

where the X_i are the elements in the regression vector, or input layer, and δ is the error in the prediction. The learning rule is identical to the algorithm used in the general least squares parameter estimation method. The linear neural network estimator shown in Figure 9 is a parallel distributed processing implementation of the least squares parameter estimation algorithm with the parameters identical to the weights, the inputs identical to the regression vector, and the output element(s) identical to the measured output(s).

In the control phase, the outer two layers in Figure 10 become active. The weights determined in the estimation phase are frozen to provide the network with an internal model of the system. The input vector is updated by the addition of the measured value y(t) and some reference input r(t)

$$\phi(t) = [r(t)\ u(t-1)\ u(t-2)\ -y(t)\ -y(t-1)\ -y(t-2)] \quad (4.4)$$

This input is then combined in some weighted sum to create an input u(t) which is physically applied to the system

$$u(t) = \sum_j w_{1j}\, \phi_j(t) \quad (4.5)$$

where the $\phi_j(t)$ are the elements in $\phi(t)$. With the exception of r(t), the other regressors in equation (4.4) are fed forward to create a predicted regression vector as seen in the second layer in Figure 10. This predicted input is propagated through the fixed model to determine a predicted value for y(t+1) in the third layer. This value, $\hat{y}(t+1)$, is fed forward through a fixed weight of one and compared to the model reference value for time t+1. The error

between the prediction, $\hat{y}(t+1)$, and the reference value is backpropagated through fixed weights until it reaches the control input u(t), where it corrects the weights, or controller gains, between the first layer and u(t). A new measurement of the output resulting from the input of u(t) determined from equation (4.5) is then made and the estimation process is repeated. The controller gains for this network are determined in some least squares sense from the error between the network predicted output and the model reference output.

In this chapter, the concepts of estimation and control are represented in two natural structures for the study of neural networks in adaptive control. Through the use of linear activation functions it has been shown that a neural network adaptive controller is a parallel distributed processing implementation of the general controller and linear parameter estimator developed in Chapter III. Each structures has its own strengths and weaknesses. The parallel structure offers more flexibility while the sequential structure uses less elements. In following chapters, these structures will be applied to specific adaptive control problems.

V. EXPERIMENTAL SETUP

Before considering specific applications of these neural network adaptive control structures, a number of experimental setup considerations must be discussed. First to be described is the computing platform used in the investigations. Next the system to be controlled and identified, the longitudinal motion of the A-4, will be introduced followed by consideration of how this system is effected by a number of adaptive control design issues. Important control design concepts include controllability and observability, stability and tracking performance. Estimation design concepts include input-output selection and model selection. Finally, a discussion of how to determine the validity of the model or controller developed. An understanding of these concepts is necessary to provide a frame of reference from which to evaluate the networks demonstrated in the next chapter.

A. HARDWARE-SOFTWARE PACKAGE

Research for this thesis was conducted on a Sun Microsystems, Inc. Sun 386i/250 workstation using the Neuralworks Professional II software package by Neuralware, Inc. No true parallel distributed processors are commercially available, so it is necessary to simulate neural network operation using software and high speed centralized processors. The requirements for hardware include large memory capacity and high speed to adequately emulate a parallel distributed processor. Software requirements include an open architecture, provision for multiple network types, flexible input-output, ease of network modification, user-friendliness, size, and speed. The combination of the Sun and Neuralworks have proven to be the best system to meet these requirements.

The Sun 386i/250 is an 80386 32-bit processor workstation operating at 25 MHz to produce performance in the range of five million instructions per second (MIPS). The system used for this investigation was configured with 16 MB of memory, one $3\frac{1}{2}$ inch floppy drive, a $\frac{1}{4}$ inch tape drive, VGA adapter, and a 16 inch monitor. With the use of Neuralworks software, the Sun is capable of generating networks with up to 20,000 elements and 1,500,000

connections operating at a rate of 45,000 connections per second. The flexibility of the Sun's operating system proved to be as important as the speed and memory size. SunOS provides a multitasking windowed graphical environment on top of the powerful UNIX operating system. In addition through the use of DOS windows, SunOS allows multitasking using DOS applications as well. This provided the capability to train multiple networks at the same time data manipulation was being conducted using both UNIX and DOS programs. No practical neural network research may be conducted without the memory capacity and speed of the Sun/386i. For the same price as a similarly configured IBM-compatible 80386 machine, the Sun offers more power and flexibility. [Ref. 9]

Hardware power and flexibility are useless without equally powerful and flexible software. The Neuralworks Professional II neural network development system by Neuralware, Inc. offers the required flexibility and power. The complexity of applications offered in Neuralworks ranges from fully developed example networks, to instant generation of standard network types, to user customization of networks at the elemental level. Nearly two dozen standard network types are available. Input-output may come from keyboard input, formatted ASCII files, various spreadsheet formats, or user defined modules written in the C programming language. Network structures may be saved in ASCII for portability between systems. The ability to perform network diagnostics and monitor internal network dynamics is provided by the use of 'probes' and 'instruments'. These software constructs may be used to graphically display or store for future use a number of important network parameters during the training process. Neuralworks allows customization of the network topology, neurodynaimcs, and network control strategy. Neuralworks Professional II is a very powerful flexible neural network development system. [Ref. 10]

Neuralworks accomplishes this flexibility through the interaction of a number of basic modules. The main executable program contains code for the generation and operation of a network, including processing element definition, learning rules, activation functions, and a number of utilities. The architecture of a network, once created, may be saved and later retrieved using a network data file. The example networks which come with the package are

contained in network data files. Input-output may be provided to the network in different ways. The main executable module contains a utility for inputing data from a number of different spreadsheet types. Complementing this is the ability to write an executable module called USERIO in the C programming language to generate data. Neuralworks provides a built-in USERIO program to input data from formatted ASCII files. To control the sequencing of input-output, propagation, and learning, Neuralworks employs user definable control strategies. Default strategies are provided for all of the standard network types. Other modules which interface with the main executable include data files for gain schedules, style sheets, and output data. Prototypes for the control strategies and USERIO programs used for this investigation are included in Appendix A. [Ref. 3]

Another software package used heavily in this research is the Pro-MATLAB interactive scientific and engineering program by the Mathworks, Inc. This program was used for all of the data generation, processing, and display used for this thesis. Written in C, MATLAB provides easy access to software developed by LINPACK and EISPACK, as well as graphics, programmable macros, IEEE arithmetic and numerous signal processing and control subroutines. Without MATLAB, the data processing requirements for this investigation would have become extremely tedious. [Ref. 11]

The Sun 386i/250 and Neuralworks Professional II combination provide an outstanding testbed for the study of any neural networks applications. The combination of speed, flexibility, power, and a user friendly environment make this combination just about the best for the study of neural networks. Also worthy of note is the contribution provided by MATLAB by Mathworks, Inc.

B. LONGITUDINAL MOTION OF THE A-4 AIRCRAFT

The system selected for use in this investigation is a simulation of the longitudinal motion of the A-4 aircraft. The complexity of aircraft motion is discussed at length in [Ref. 12] and [Ref. 13]. The system and some of its important characteristics, as well as the manner in which it was simulated will be briefly described below. Through the use of a

number of assumptions, the most significant of which is the assumption of small perturbations [Ref. 12:pp. 84-127], the complex motion of an aircraft may be reduced to uncoupled sets of linear equation for lateral and longitudinal motion at specified flight conditions. The longitudinal equations of motion may be expressed in state space form as

$$\dot{\underline{x}}(t) = A\underline{x}(t) + B\underline{u}(t) \tag{5.1}$$

$$\underline{y}(t) = C\underline{x}(t) + D\,\underline{u}(t)$$

where the state variable is defined by

$$\underline{x}(t) = \begin{vmatrix} u(t) \text{ -- airspeed perturbation} \\ \alpha(t) \text{ -- angle of attack perturbation} \\ q(t) \text{ -- pitch rate perturbation} \\ \theta(t) \text{ -- pitch angle perturbation} \end{vmatrix} \tag{5.2}$$

the input variable is defined as

$$\underline{u}(t) = \delta(t) \text{ -- elevator deflection} \tag{5.3}$$

and the output variables are scaled versions of the state variables. The A and B matrices are constructed for a given flight condition (altitude and mach number) from the aircraft's non-dimensional stability derivatives, mass, moments of inertia, altitude and airspeed. The C matrix is a diagonal scaling matrix, and the D matrix is the zero matrix. Descriptions of this process are contained in [Ref.12:pp. 167-196] and [Ref. 13:pp. 112-144].

This system was selected for this application because it exhibits a number of interesting characteristics. It is a higher order, multiple output system. The natural time constants of the system are of different orders of magnitude. The response of the variables u(t) and θ(t) is dominated by low frequency dynamics, while the response of α(t) and q(t) is predominately high frequency. The frequency response for the aircraft at sea level and mach 0.4 is shown in Figure 11. The low frequency, or phugoid, modes have a natural frequency on the order of 0.015 hertz for a time constant of approximately 66 seconds. The high frequency, or short period, modes have a natural frequency on the order of 0.5 hertz for a time constant of 2 seconds. By using several linear models for different flight conditions in the simulation,

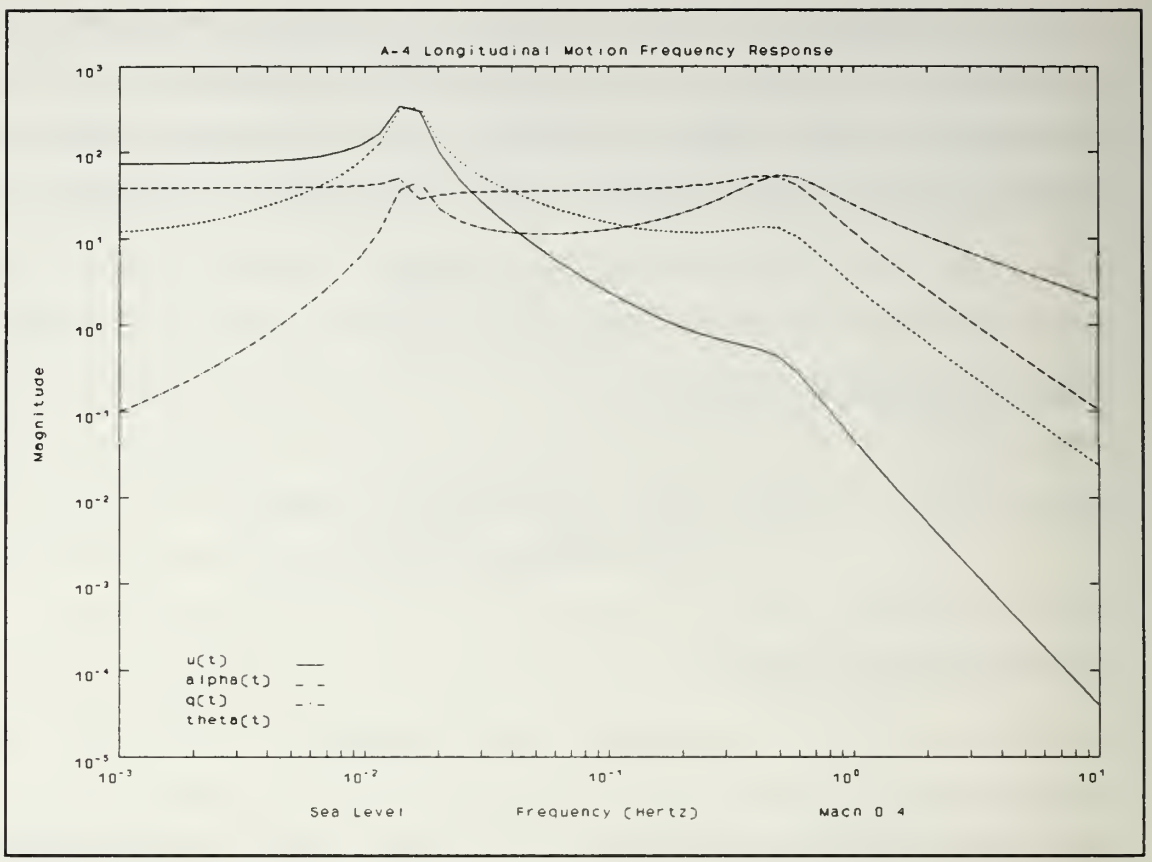

Figure 11: A-4 Frequency Response

a crude form of non-linearity may be introduced. The five different linear models depicted in Table I were used for this investigation. The manner in which these system characteristics effect the adaptive control process will be addressed after the simulation process is described.

Table I: Flight Conditions Selected for Study

Flight Condition	Altitude	Mach Number
Condition 1	Sea Level	Mach 0.4
Condition 2	15,000 ft	Mach 0.5
Condition 3	35,000 ft	Mach 0.6
Condition 4	35,000 ft	Mach 0.8
Condition 5	Sea Level	Mach 0.8

The simulation was carried out through the use of a recursive algorithm in the USERIO program. Continuous state space models (5.1) for each of the five conditions were developed using data from [Ref. 13:Appendix II] and converted to difference equations (3.3) for each state. A sample of the MATLAB script file used to generate this data is included in Appendix B. The values used for the continuous state space models and discrete transfer functions are given in Appendix C. Further description of the computations performed may be found in the Pro-MATLAB reference manual [Ref. 11]. The process involved first scaling the states using the C matrix then developing a balanced realization to ensure better conditioned matrices. These matrices were then converted from a continuous state space to a discrete state space model using a matrix polynomial algorithm. This discrete state space model was then converted to a discrete matrix polynomial transfer function using

$$H(z) = C\,(zI - A)^{-1}\,B = Y(z)/U(z) \tag{5.4}$$

The z-transform may be replace with the q^{-1} operator and the resulting function may be divided into numerator and denominator terms to obtain the DARMA model equation

$$A(q)\,\underline{y}(t) = B(q)\,u(t) \tag{5.5}$$

or by a simple rearrangement

$$\underline{y}(t) = B(q)\,u(t) - (A(q) - 1)\,\underline{y}(t) \tag{5.6}$$

or, upon expansion of the matrix polynomials

$$\underline{y}(t) = \begin{vmatrix} b_{u1}q^{-1} + b_{u2}q^{-2} + b_{u3}q^{-3} + b_{u4}q^{-4} \\ b_{\alpha 1}q^{-1} + b_{\alpha 2}q^{-2} + b_{\alpha 3}q^{-3} + b_{\alpha 4}q^{-4} \\ b_{q1}q^{-1} + b_{q2}q^{-2} + b_{q3}q^{-3} + b_{q4}q^{-4} \\ b_{\theta 1}q^{-1} + b_{\theta 2}q^{-2} + b_{\theta 3}q^{-3} + b_{\theta 4}q^{-4} \end{vmatrix} u(t) - [\, a_1q^{-1} + a_2q^{-2} + a_3q^{-3} + a_4q^{-4} \,]\ \underline{y}(t) \tag{5.7}$$

The delay operator terms in this system of equations are then expanded and the equations are rearranged to obtain four separate recursive equations of the form

$$y_j(t) = \sum_i [b_{ji}\,u(t-i)] - \sum_i [a_i\,y_j(t-i)] \tag{5.7}$$

where the y_j terms indicate the outputs u(t), $\alpha(t)$, q(t), and $\Theta(t)$ and the four a_i terms are duplicated for each of the four equations. The 20 parameters from equation (5.6) along with

equation (5.7) combine to make the algorithm used in the USERIO program to recursively simulate the longitudinal motion of the A-4 aircraft. The frequency response of this discrete simulation for the condition at Mach 0.4 and Sea Level with a sampling time of 0.1 seconds is given in Figure 12. The coefficients for the B(q) and (A(q) - 1) matrices for this condition

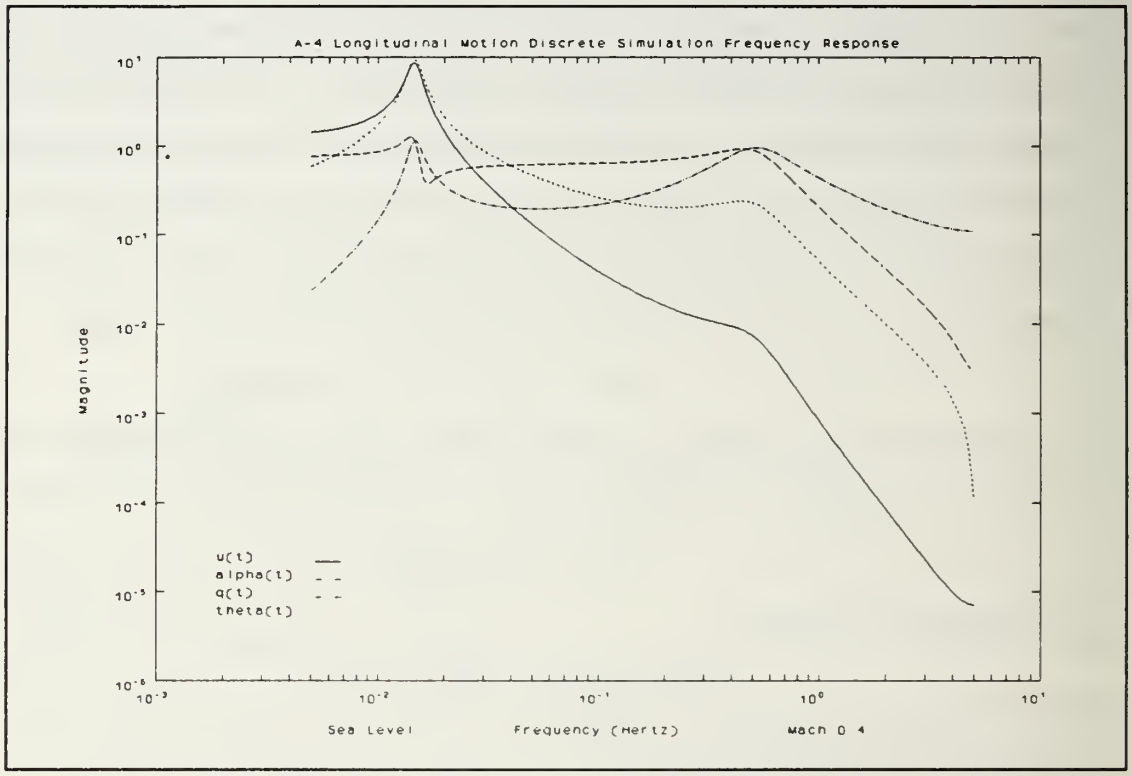

Figure 12: Frequency Response for Discrete A-4 Longitudinal Motion Simulation

and sampling rate are given in Table II where the position of the coefficients match those in equations (5.6) and (5.7). These coefficients are relatively numerically ill-conditioned. With the six decimal place precision of the Neuralworks Professional II software it will be difficult to characterize the u(t) terms in B(q) (the first row in Table II) which are on the order of 10^{-5}. The same is true to a lesser extent for the other terms in B(q). At the same time, there is a large difference in relative magnitude between the (A(q) - 1) terms and the B(q) terms for each output. The ill-conditioned nature of this problem can make it difficult to determine a good model for the system.

Table II: Parameters for Flight Condition Sea Level/Mach 0.4 with a Sampling Time of 0.1 Seconds

B(q) =	2.7131e-05	7.7248e-05	-7.0558e-05	-2.2579e-05
	-3.4619e-02	4.5022e-02	1.3715e-02	-2.4124e-02
	-1.9864e-01	5.8002e-01	-5.6414e-01	1.8276e-01
	-7.7052e-03	7.4781e-03	6.9640e-03	-6.7386e-03
(A(q) - 1) =	-3.6949e+00	5.1802e+00	-3.2755e+00	7.9021e-01

C. EXPERIMENTAL DESIGN CONSIDERATIONS

A number of design issues must be carefully considered in the implementation of a neural network adaptive controller. These concerns are driven by the complex interrelationships between the system, the controller, and the estimator. The experimental setup is controlled in large part by these matters. Failure to address these topics may result in a failure of the neural network controller. The manner in which system, control, and estimation concerns effect the design of the neural network adaptive controller are discussed below.

1. Control Design Issues

A number of different factors must be considered in the design of a adaptive controller. In order for the estimator to function, the system must be observable. In order for the controller to work, the system must be controllable. Systems, such as the longitudinal motion of the A-4, which can be expressed as transfer functions are both controllable and observable. For this controllable, observable system, some control objective must be formulated. The neural network adaptive controller was conceived as a type of model reference adaptive controller. The USERIO module generates a model reference output using the parameters for the flight condition at 15,000 feet and mach 0.5 in parallel with the simulation. This condition was chosen because it is relatively close to the center of the flight envelope determined by the other four flight conditions. The control objective is to track this reference output. [Ref. 5:p. 152]

Stability is also an important issue. The poles and zeros of A(q) and B(q) for the simulation at Condition 1 (Table II) are given in Table III. Note that u(t) has a zero outside the unit circle, q(t) has a zero on the unit circle, and $\alpha(t)$ and $\theta(t)$ have poles very near the unit circle. These zeros are or potentially may become non-minimum phase. This will cause the inverse of the transfer function to be unstable, requiring infinite or non-causal control for exact tracking. At the same time, an offshoot of the concept of controllability is that an independent input is required to exactly control an independent output. For this system, there is a single input with four outputs. The solution to these two problems is to use some non-exact form of tracking. In the neural network adaptive controller, the non-exact tracking is handled in two ways. First, the control input activation function can be limited to a certain value. This simulates control saturation. Second, the control gains are determined in some least squares sense using the Backpropagation learning rule similar to the

Table III: Poles and Zeros of the Discrete Simulation for Condition 1 with a Sampling Time of 0.1 Seconds

poles =	0.8482 + 0.2681i
	0.8482 - 0.2681i
	0.9993 + 0.0096i
	0.9993 - 0.0096i
$zeros_{u(t)}$ =	-3.5191
	0.9270
	-0.2551
$zeros_{\alpha(t)}$ =	0.9992 + 0.0100i
	0.9992 - 0.0100i
	-0.6979
$zeros_{q(t)}$ =	1.0000
	0.9986
	0.9213
$zeros_{\theta(t)}$ =	0.9986
	0.9219
	-0.9500

method used by conventional optimal control. The purpose of optimal control is to achieve the best possible non-exact tracking given certain constraints in a least squares sense. Through the use of a type of optimal control and control saturation, the neural network adaptive controller should be unaffected by the presence of unstable inverses in the system. [Ref. 5:pp. 157-163]

2. Estimation Design Issues

Although based on the same principles, the estimation design considerations for this investigation are more complex than those for control. The goal of an estimator is to develop a model of a system for a specified purpose. For use in the neural network adaptive controller, the function of the estimator is to model the input-output relationships of the system. To accomplish this objective, determination of appropriate input-output characteristics and model structure must be made. These design decisions must be tempered by consideration for the model application--a neural network adaptive controller.

a. Input-Output Selection

In estimation experiments, the selection of what to measure is a complex issue. In this case the state and input variables are the measured outputs and input, however the scaling of these measurements is an important factor which will be described in the discussion of model structure. Once the variables to be used are selected, the proper input characteristics for the experiment must be determined. Three factors which must be considered in input selection are data record length, the input spectrum, and the sampling time. [Ref. 6:p. 340]

The choice of input spectrum is one of the most important in estimation. Intuitively, the input spectrum must be selected such that all modes of the system are excited. This is known as the concept of persistent excitation. A related concept, parameter sensitivity is the sensitivity of the parameters to excitation at different frequencies. This is a function of not only the system to be modelled but the model structure chosen as well. There are disadvantages to overexcitation, however. The input spectrum must not be selected in such a manner that the output signal strength is exceeded by any expected non-modelled

noise. This signal to noise ratio concept is related to the idea of the information content of an input. [Ref. 6:pp. 358-378]

The effects of these factors in input spectrum selection may be seen graphically in Figure 12. Persistency of excitation indicates that at a minimum input energy must be placed near the low frequency, or phugoid modes, and the high frequency, or short period modes. Due to sensitivity of parameters, energy must also be expended in the range of frequencies where high frequency attenuation occurs. The justification for this may be best understood by recalling that the best indicator of a system's order is its high frequency roll-off. The concept of a truly deterministic system is impossible to obtain in real terms. In this, as in all, investigation of deterministic systems there is actually some noise present. This noise is due primarily to simulation errors and truncation in the Neuralworks Professional II program (Neuralworks only allows access to values with precision out to six decimal places.) Both of these noise factors are predominantly in the high frequencies where it was just indicated that there must be some excitation. The presence of modelling errors, known as aliasing, may be seen by comparing the high frequency (three to five hertz) regions in Figure 12, the system discrete simulation frequency response, with Figure 11, the true system frequency response. Note that there are some high frequency dynamics present in the discrete frequency response which are not present in the continuous, or true, frequency response. The concept of information content thus is in conflict with the concept of persistent excitation. The problem of input spectrum is to select the input which is the best compromise between the requirements of persistent excitation and information content. The complex issues in input selection are studied in this investigation through the use of a number of different user selectable inputs in the USERIO program. [Ref. 6:pp. 358-378]

Another factor in input-output selection is determination of the sampling time. Sampling a system contaminates its dynamics. Information about frequencies above the Nyquist frequency (one half of the sampling rate) is totally lost. At the same time, energy in frequencies above the Nyquist frequency is folded over onto lower frequencies. This superposition is the aliasing described in the previous paragraph. The desire is to minimize

the effects of aliasing. This can be done accomplished by filtering out the aliasing and including the filter structure in the estimator. Since the structure of the filter would be known it could easily be included in the neural network structure proposed in this thesis This method adds undesired complexity to the problem and was not investigated. An alternative solution is to sample fast enough to eliminate the effects of aliasing. This, however, has its own disadvantages. [Ref. 6:pp. 378-386]

Sampling too fast may cause loss of information on low frequency dynamics while sampling too slow may cause loss of information on high frequency dynamics. The problems with sampling rates result from poor numerical conditioning, aliasing, and the distribution of energy in the input spectrum. Figure 13 shows the pole-zero plots which result from sampling the system of A-4 longitudinal motion at two different sampling rates. With a

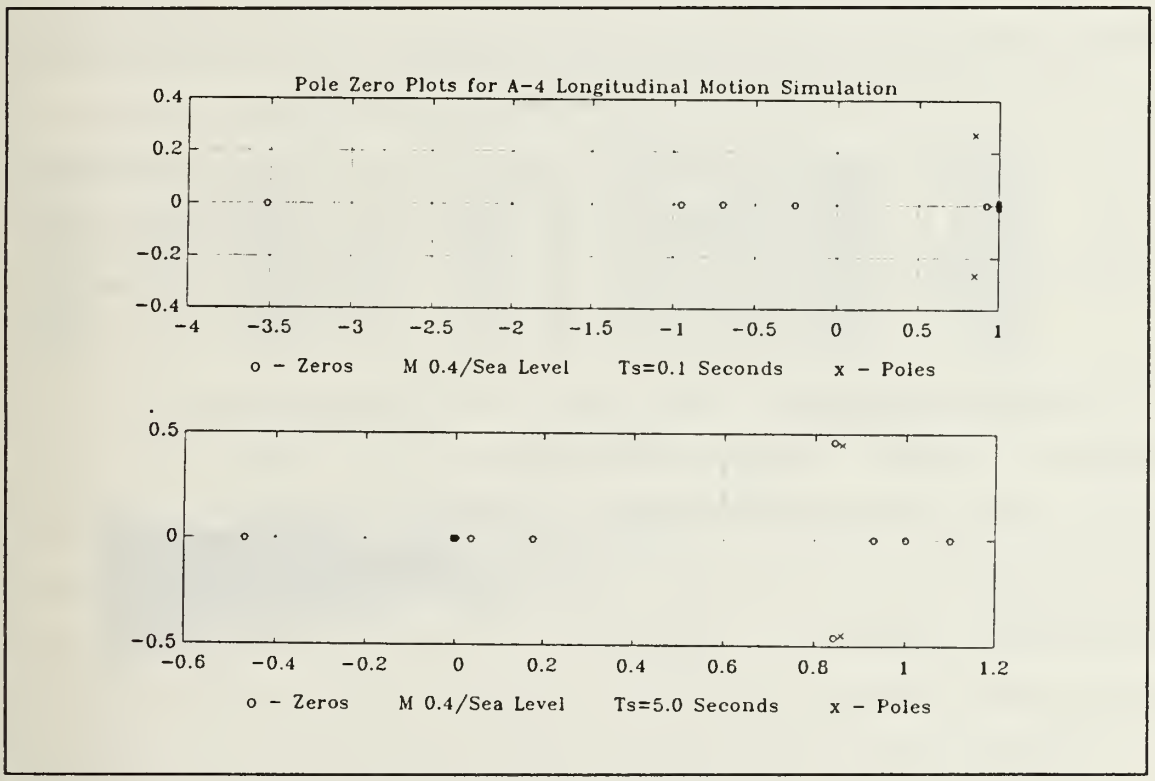

Figure 13: The Effect of Sampling Rate on Poles and Zeros

sampling time of 0.1 seconds, the low frequency poles and zeros congregate around the point z = 1.0, the ideal integrator. With a sampling time of 5.0 seconds, the high frequency poles

cluster around the point z=0.0, a direct input. Sampling too fast may cause ill-conditioning in the low frequencies while sampling too slow may cause ill-conditioning in the high frequencies. Another problem with sampling too slow results from the aliasing discussed in the previous paragraph, while another problem which is a consequence of sampling too fast is the energy distribution problem demonstrated in Figure 14 where the input sequence and spectrum for a random binary (RB) input are shown. By using a log log plot, it is easy to see that each succeeding decade of the input spectrum contains ten times more data points implying ten times the excitation and thus ten times the spectral energy. The higher frequency modes therefore receive more excitation. The consequences of fast and slow sampling rates indicate that estimators will only be effective over a limited range of frequencies. An estimator can successfully cover on the order of two to three decades of

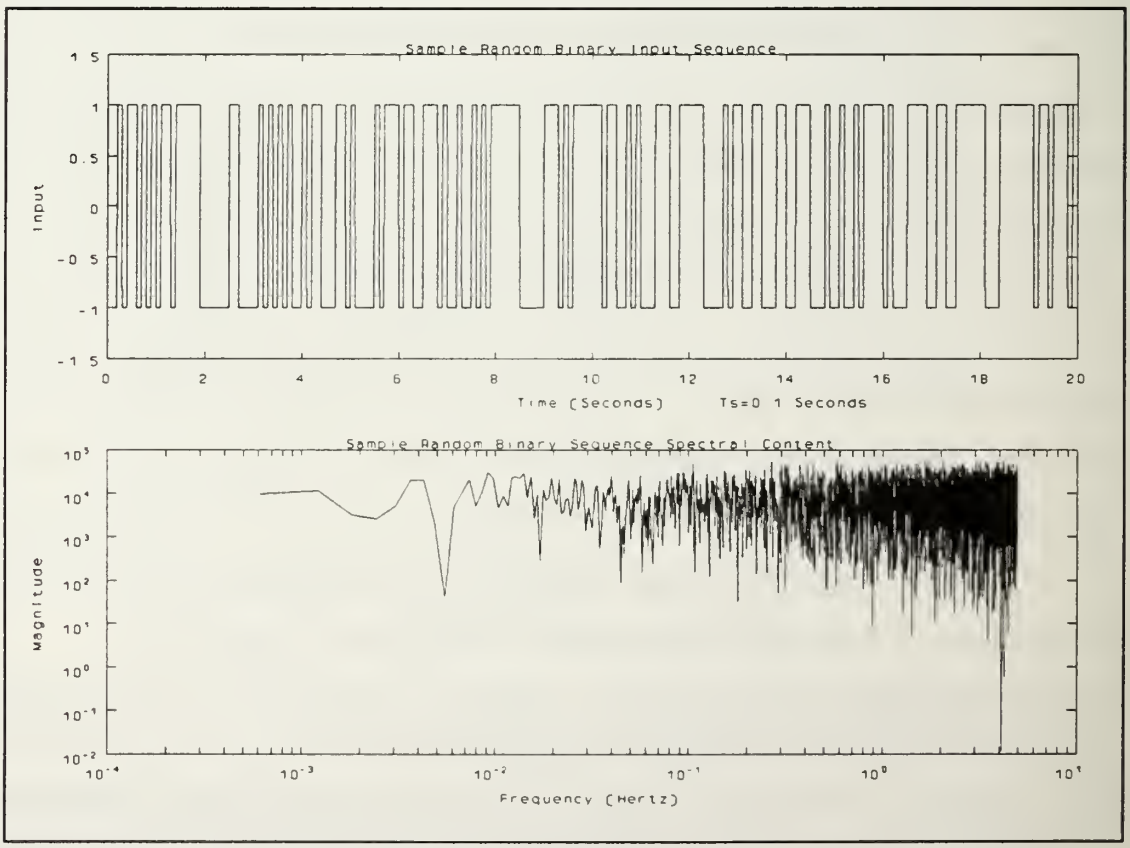

Figure 14: Effect of Sampling Rate on Excitation

frequencies. A rule of thumb for the selection of sampling time is approximately one-tenth of the highest natural time constant. For this investigation, a sampling times of 0.1 seconds was used. The low frequency and high frequency dynamics of the A-4 longitudinal motion span a range of about two and one half decades and thus may exceed the limitations of the estimation process. [Ref. 6:pp. 378-386]

The choice of data record length is also important. Although the simulation used in the USERIO program could generate data indefinitely, modeling errors result from the recursive nature of the simulation and the presence of non-minimum phase zeros. Errors in the simulation propagate at a rate proportional to the power of the absolute value of the system zeros. Since many of the zeros are near to or outside of the unit circle (see Table III) errors in the simulation will grow unbounded. The simulation must thus be reset at some time to keep these errors from becoming significant. At the same time, resetting the simulation adds noise to the data by truncating the sequence. There is also a need to consider the number of periods of the phugoid modes presented to the estimator. The chosen compromise was to use a data record length of 9000 points which would give between 125 to 150 presentations of the phugoid using a sampling time of 0.1 seconds. This is implemented in the USERIO program by resetting the simulation every 9000 cycles. [Ref. 6:p. 382]

Input-output selection is a complex task. Consideration of persistency of excitation, sensitivity of parameters, and information content is essential in selecting the input spectrum. The range of significant frequencies in the system is important in choosing a sampling rate. Finally, the length of the data record must be commensurate with the size of the model and character of the system being modelled. The USERIO program used for this investigation incorporates a number of input selections to test the effects of some of these choices, while minor modifications to the program may be used to test others.

b. Model Selection

Model selection is also critical to the success of the estimation process. This should not be viewed as the selection of a single model, but instead the selection of a class of models. The estimation process is the determination of which member of this class best fits the data provided. In this way, the model selection represents some artificial constraint in which the system is to be represented. The major factors in model selection include model structure, parameterization, and the estimation algorithm. The model structure represents the architecture of the model while the parameterization determines the dependencies of its elements and the estimation algorithm determines the manner in which the dependencies are changed. Model selection issues are important to consider in the design of a neural network adaptive controller.

Selecting the model structure involves choosing a prototype for the system. This may include determining whether the model is to be linear or non-linear, the order of the system, the number of elements in the input and output vectors, even the number of models used to represent the system. For the adaptive controller, the model structure also includes the number of elements used for the control law. For this investigation, the model structure was similar to the sequential structure developed in Chapter IV. Figure 15 gives an example of the structure used parameterized as four different transfer functions. Since the longitudinal motion of the A-4 is a fourth order system with four outputs, the required number of regressors is 20, four for each output and four for the single input. The 19 elements in the bottom layer represent past measurements of the outputs and inputs. From left to right, the first three input elements are $\delta(t-2)$, $\delta(t-3)$, and $\delta(t-4)$ where the element label indicates the delay for that particular unit. The next four elements represent delayed values, or past measurements, of u(t), followed by four for $\alpha(t)$, four for q(t), and finally four for $\Theta(t)$. This layer is duplicated in the second layer with the addition of r(t-1), the reference input, to provide the state variable plus reference input for the control law synthesis. The third layer is a single element, a weighted sum of the reference input and states, the control input, $\delta(t-1)$. The 20 elements in the regression vector come from the 19

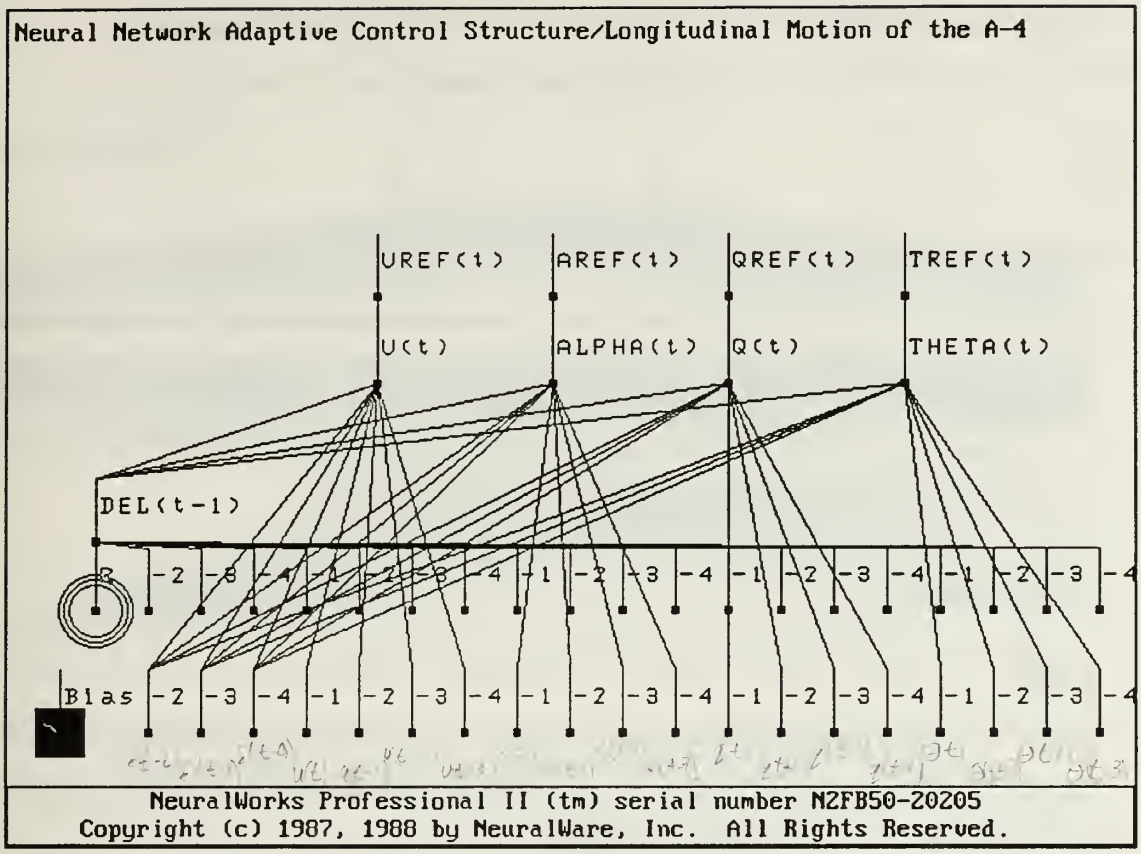

Figure 15: Neural Network Adaptive Controller Structure for A-4 Longitudinal Motion

units in the first layer and the single unit in the third layer, which is itself a weighted sum of the activations of the elements in the second layer. This is a slightly different, though equivalent, structure for the middle layers than the one developed in Chapter IV. The second layer from the top is the output layer. The top layer is the reference layer. The effects of non-linear models are incorporated by introducing hidden layers with non-linear activation functions between the third layer, the control input layer, and the fourth layer, the output layer, in Figure 15. Additional elements may also be added to the input vector to allow for models of higher order or different inputs as in Figure 16. Note the addition of mach number and altitude in the last eight elements in the bottom layer. This neural network adaptive control structure allows a natural progression from linear to nonlinear models. Determining the manner in which these elements of the model structure are connected is known as parameterization. [Ref. 6:pp. 408-431]

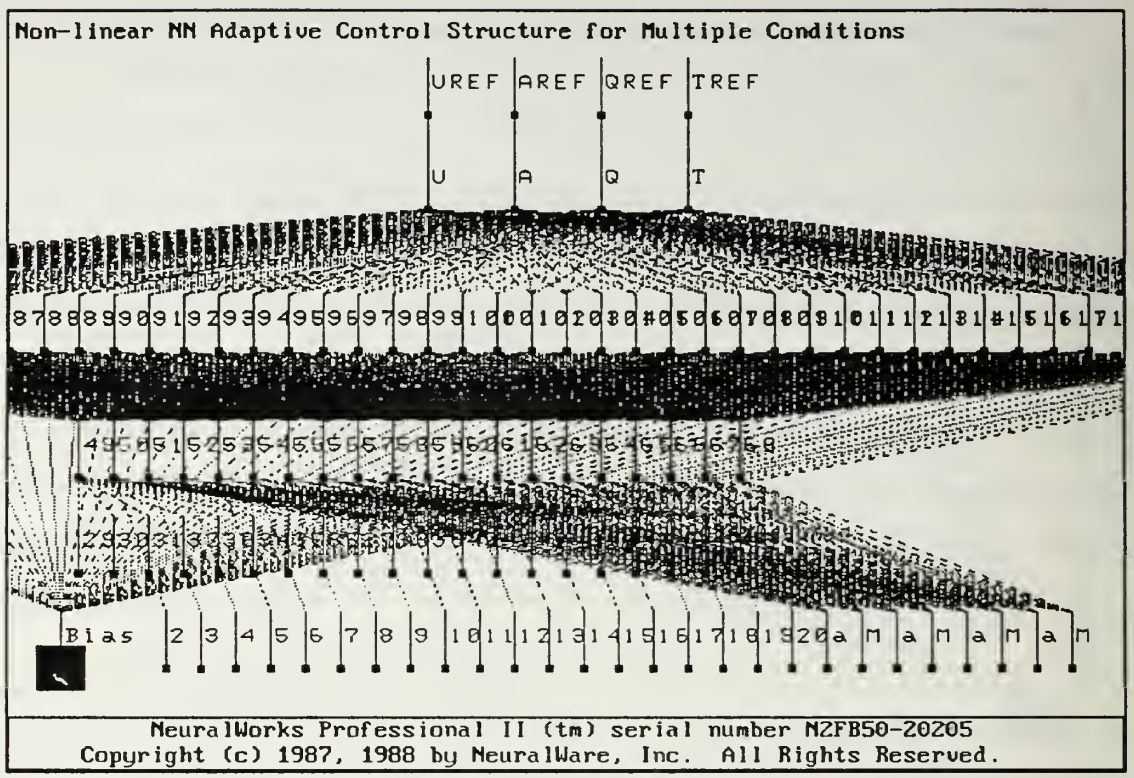

Figure 16: Non-linear Neural Network Adaptive Controller Structure for A-4 Longitudinal Motion/Multiple Conditions

Parameterization concerns selecting the dependencies between various elements in the estimator. From equation (5.7), it is known that it requires a minimum of 20 parameters to fully describe the system. From Figure 15 it can be seen that there are 80 possible connections in the linear structure which can be made in any number of combinations. The neural network adaptive control structure in Figure 15 is parameterized as four separate linear transfer functions. This requires 32 connections. Each output element has eight connections. Four of these connect the $\delta(t)$ terms to the output element and represent the b_i parameters from equations (5.6) and (5.7). The other four connections attach the output to its own past values and represent the a_i parameters in (A(q) - 1) from equations (5.6) and (5.7). More elements may be added to the regressor to change the order of the model. The selection from all possible parameterizations is a complex task which will be discussed in the results. [Ref. 6:pp. 408-431]

The complexity of the gain used in the estimation algorithm has a great effect on convergence and stability of the estimator. The gain term in the general form of the least squares estimator

$$\Theta(t+1) = \Theta(t) + M(t)\,\phi(t)\,e(t) \tag{5.8}$$

may take many forms, one of the most complex of which comes from the RLS estimation method developed in Chapter III. The gain used in the Backpropagation neural network is much simpler. It is related to an estimation algorithm known as the least mean square (LMS) estimation method, whose gain law is

$$M = \frac{\alpha}{\| \Phi \|} \tag{5.9}$$

where α is some constant and $\| \Phi \|$ is some norm of the entire set of regression vectors. The stability limits for α result from the fact that the linear estimation process is itself a first order dynamic system. The value of α in this system represents the eigenvalue of the system whose stability limits are [Ref. 14]

$$\begin{array}{ll} 0 < \alpha < 1 & \text{Stable, overdamped} \\ 1 < \alpha < 2 & \text{Stable, underdamped} \\ 2 < \alpha & \text{Unstable} \end{array} \tag{5.10}$$

Knowledge of these limits on α gives an exact means to determine the value for the learning rate in a linear Backpropagation neural network and a general guideline for determination of the value for the learning rate in a nonlinear Backpropagation neural network. An analysis of the features provided by the gain term in the RLS method will point out some other ways in which the Backpropagation estimation gain may be improved. The RLS gain term is expressed as

$$L(t) = \frac{P(t)}{1 + \phi^{T}(t)\,P(t)\,\phi(t)} \tag{5.11}$$

Of no small significance is the fact that the gain is a function of time. In both the numerator and denominator, the term P(t) is the error covariance of $\Theta(t)$. It provides an error distribution function. In the denominator, the $\phi^{T}(t)\,P(t)\,\phi(t)$ term provides some scaling

function. Error assignment in distributed systems is a complex task beyond the scope of this investigation. The gain can, however, be made more robust by making it a function of time and prescaling the data. A crude form of adaptive gain law is provided through the use of gain schedules, known in the Neuralworks Professional II software as learning rate schedules. The use of learning rate schedules is at best a trial and error effort. They were not used in this investigation. The data may also be scaled to make the estimator equally sensitive to outputs of different orders of magnitude. The A-4 longitudinal modes were scaled using the C matrix as mentioned above to obtain inputs and outputs bounded by the value one. This was done empirically by simulating the system response to various inputs and scaling by the maximum deflections, at best an inexact method. The concept of scaling is also important for nonlinear activation functions whose values are bounded by set regions. Using this type of scaling, the gain for an element using the stability limits for the LMS method can be simply expressed as

$$M = 1/N \tag{5.12}$$

where N is the number of input connections to that particular element. The neural network is very sensitive to this gain. A value for the gain which is too high will cause the estimation network to go unstable. A value which is too low will require very long convergence times. The estimator gain used in the Backpropagation neural network leaves much to be desired.

3. Validation Issues

Once a model is established or a control law is developed, some means must be used to validate the result. In this investigation, the true system is fully specified, so it is easy to compare the estimated models to the true model. The neural network estimator or controller performance can be evaluated dynamically by examining the errors between the network output and the desired output as training progresses. Statically, the performance may be evaluated in the time domain by looking at the error produced using inputs other than the one on which the network was trained. Frequency domain characteristics for black box models may be evaluated using spectral estimation techniques. For linear systems, where

the parameters have some physical significance, the parameters may be used for evaluation. The coefficients of true and estimated models may be compared directly, frequency response plots may be generated, and the poles and zeros may be evaluated. Many of these methods will be used to evaluate the neural network adaptive controllers presented in the next chapter.

D. SUMMARY OF EXPERIMENTAL SETUP

In this chapter, the experimental setup for this investigation has been characterized. The hardware and software to be used were described. The system to be modelled and controlled has been introduced. Finally, some considerations in the design of the experiment were developed. With due consideration of all of the items discussed in this and the previous chapters, it is now possible to conduct experiments in the use of neural networks in adaptive control.

VI. RESULTS AND DISCUSSION

The concepts developed in the previous chapters will be combined in this chapter to demonstrate the effectiveness of the application of neural networks in adaptive control. Initially, the stability characteristics of a linear neural network adaptive control structure will be investigated. The estimation qualities of linear and nonlinear neural network adaptive control structures will then be examined. Finally, a few examples of the operation of a neural network adaptive controller will be demonstrated. Through the use of linear and nonlinear networks, the similarities between neural networks and current adaptive control techniques will be shown as well as some possible extensions of adaptive control provided by neural networks.

A. NEURAL NETWORK STABILITY CHARACTERISTICS

The neural network adaptive control structure parameterized as four different transfer functions as shown in Figure 15 was used to demonstrate network static and dynamic stability. This parameterization was chosen because the weights in the network can be directly compared to the coefficients used in the simulation. The stability demonstration was conducted by testing a network whose weights were artificially set to be exactly those of the true system. In this case, the true system was represented by the flight condition of mach number 0.4 and an altitude of Sea Level (Condition 1 from Table I). First, the network was trained for one data set, or 9000 cycles (900 seconds), using the random binary signal. Plots of the percent deviation of each of the weights, or coefficients, from the true coefficients for each output as a function of training time is shown in Figure 17 through Figure 20. Each graph contains eight plots, one for each of the a_i and b_i coefficients associated with each output. The maximum deviation of the parameters associated with u(t) is on the order of two percent (Figure 17) with one perturbation between 700 and 800 seconds, while the maximum deviation for parameters associated with $\alpha(t)$ is on the order of 0.001 percent (Figure 18) with perturbations around 400 seconds, 800 seconds, and 900 seconds. The maximum deviation

for parameters associated with q(t) is on the order of 6×10^{-5} percent with no perturbations (Figure 19), while the maximum deviation of the parameters associated with $\Theta(t)$ is on the order of 6×10^{-3} (Figure 17), also with no perturbations. At this point the discussion of the relative size of the parameters in Chapter 5, Section B. becomes apparent. Each of these percentages represents real deviations on the order of 10^{-7}, the precision of the Neuralworks Professional II program. Since the b_i parameters associated with u(t) are so small, they are very sensitive to changes in the seventh decimal place, followed in sensitivity by $\Theta(t)$, $\alpha(t)$, and finally q(t). Note that the weights and the corresponding parameters remain very stable, with few perturbations of small magnitude.

Next, the weights were each perturbed by some random amount between -0.01 and 0.01 and the network was trained for 18,000 cycles (1800 seconds). This was done to determine if the parameters would return to the original values. The percent deviation of each of the parameters as a function of training time for each output are given in Figure 21 through Figure 24. Again, the sensitivity of the parameters associated with u(t) is seen in the 500 percent deviation caused by a perturbation on the order of 0.01 in Figure 21. The parameters for u(t) appear to settle to a point near zero percent deviation. Note the underdamped convergence of the parameters for u(t). For $\alpha(t)$ (Figure 22), the parameters converge to some value within the first 200 seconds, though one parameter exhibits a deviation of approximately one percent. The same convergence rate is seen in Figure 23 for the parameters associated with q(t) with much smaller percentage deviations. The parameters for $\Theta(t)$ exhibit the same underdamped convergence seen in u(t) with percentage deviations of the same order as the parameters associated with q(t). All of the parameters show a strong tendency to return to the proper value. Deviations are again related to the relative size of the parameters for each output. The slow convergence seen in the parameters for u(t) and $\Theta(t)$ is most certainly related to the fact that u(t) and $\Theta(t)$ are slowly changing, or low frequency, modes. These two simple trials indicate that a network containing weights related to the true system will be relatively stable in the presence of small plant disturbances. At the same time, problems related to the conditioning of a model parameterized as four separate transfer functions becomes apparent.

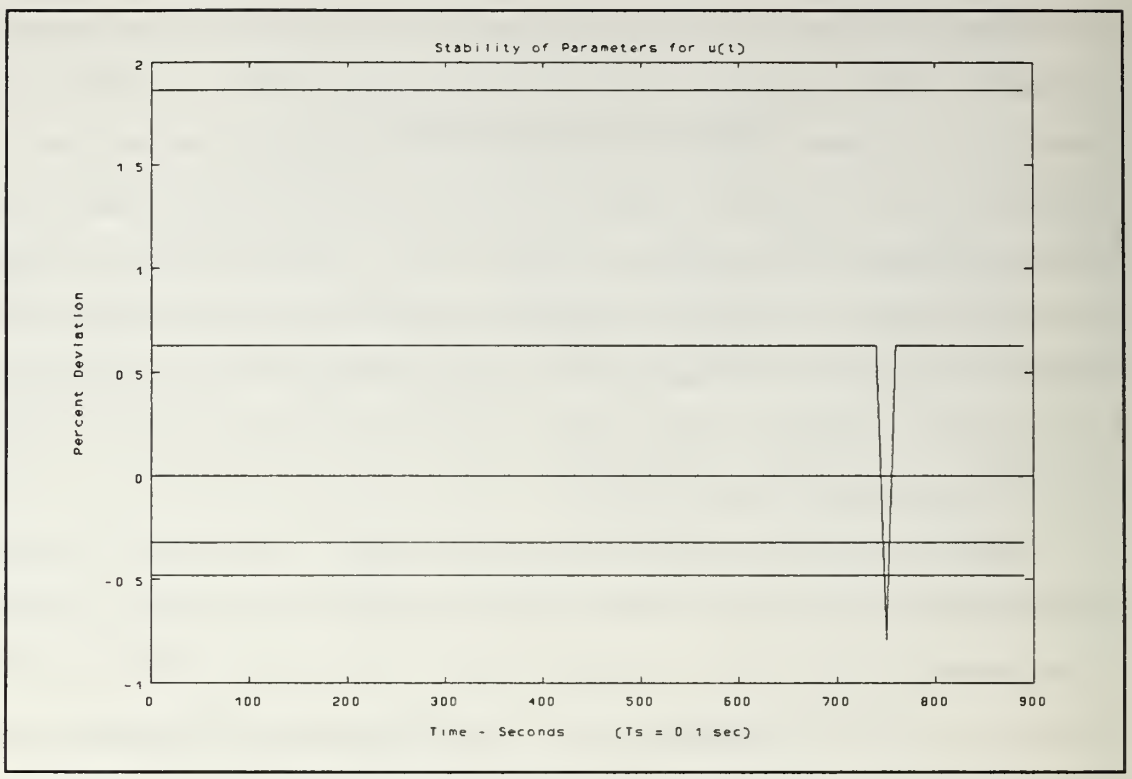

Figure 17: Network Static Stability for u(t)

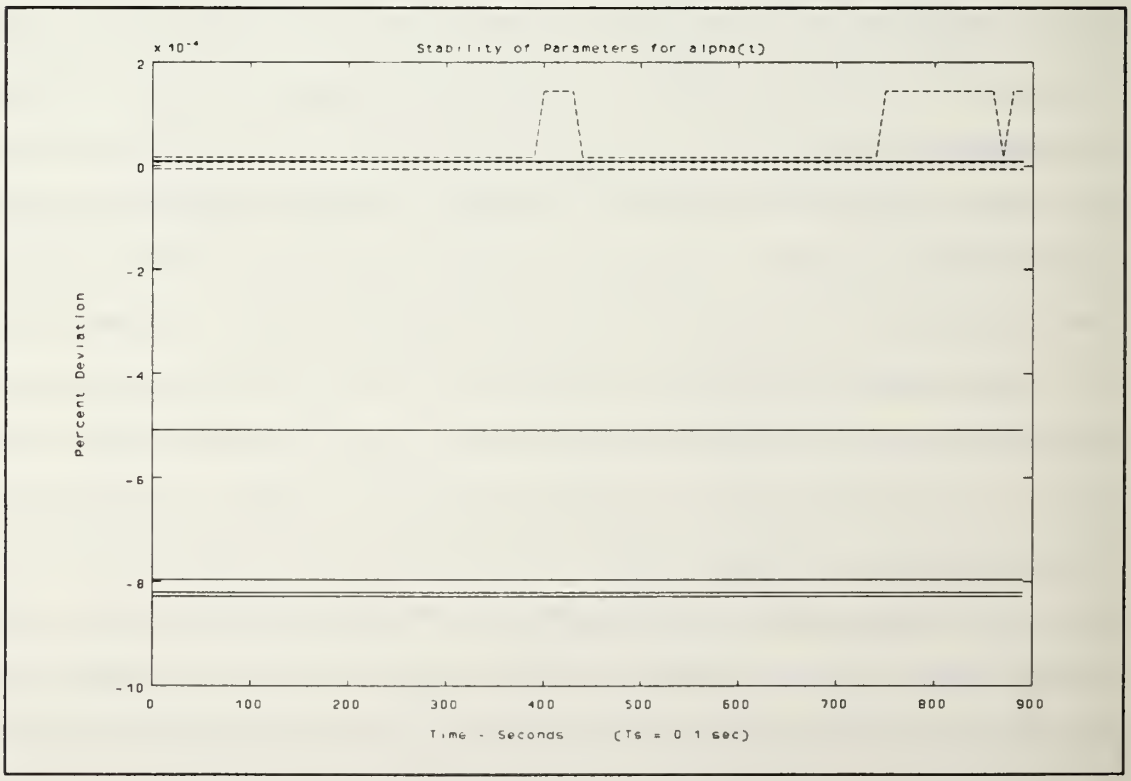

Figure 18: Network Static Stability for $\alpha(t)$

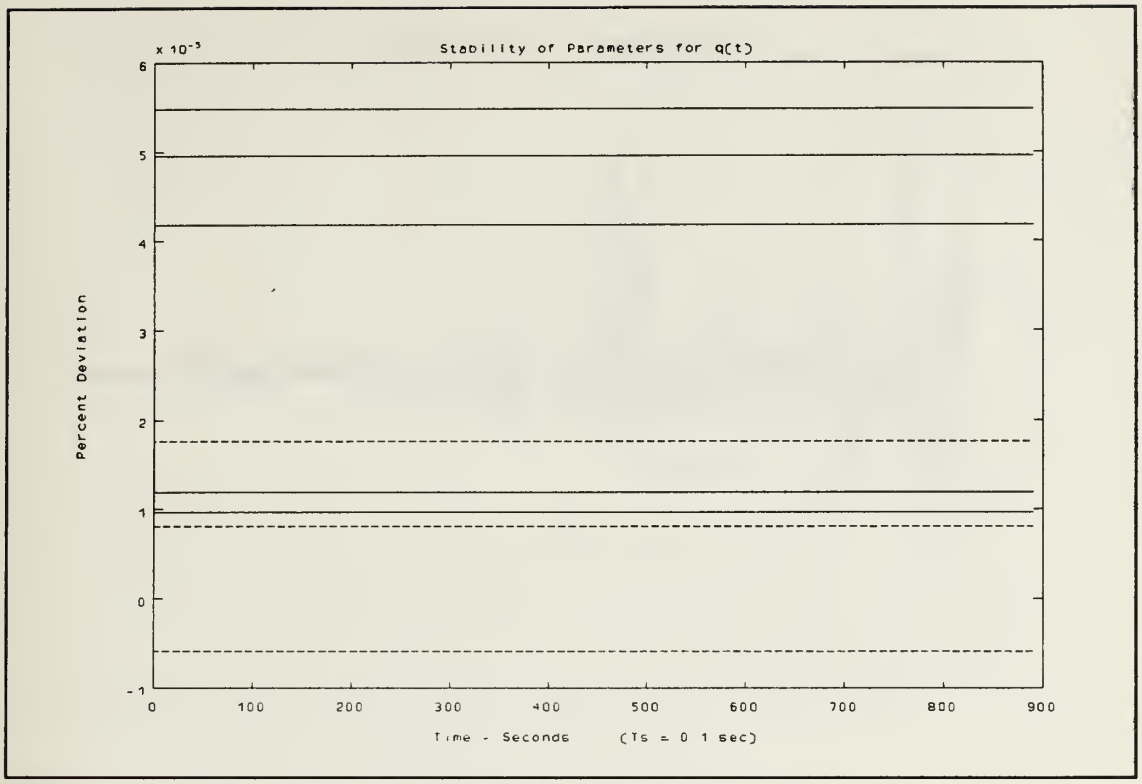

Figure 19: Network Static Stability for q(t)

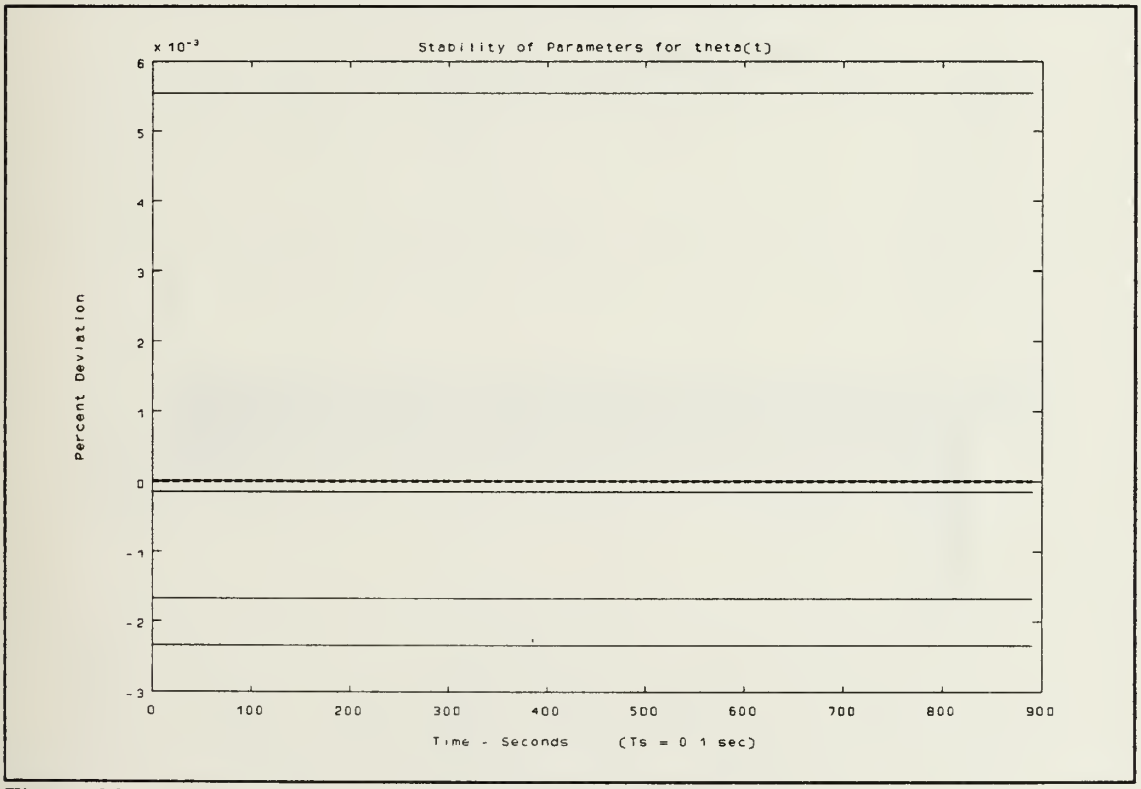

Figure 20: Network Static Stability for Θ(t)

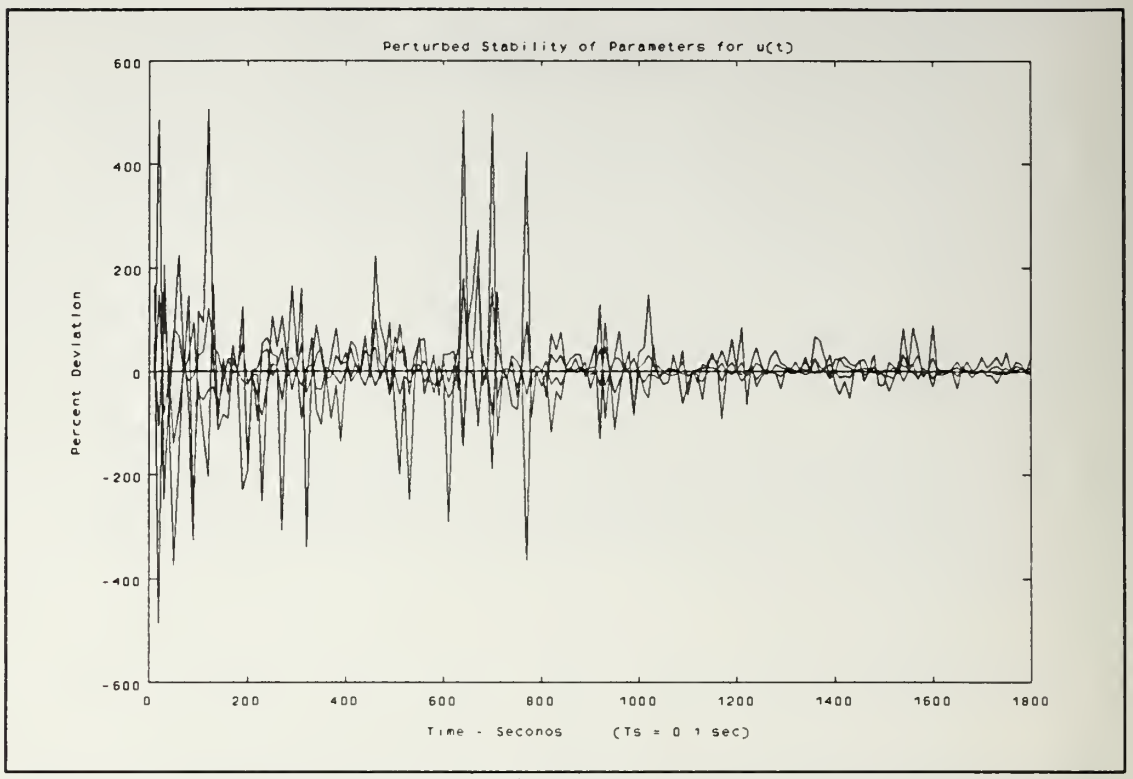

Figure 21: Network Dynamic Stability for u(t)

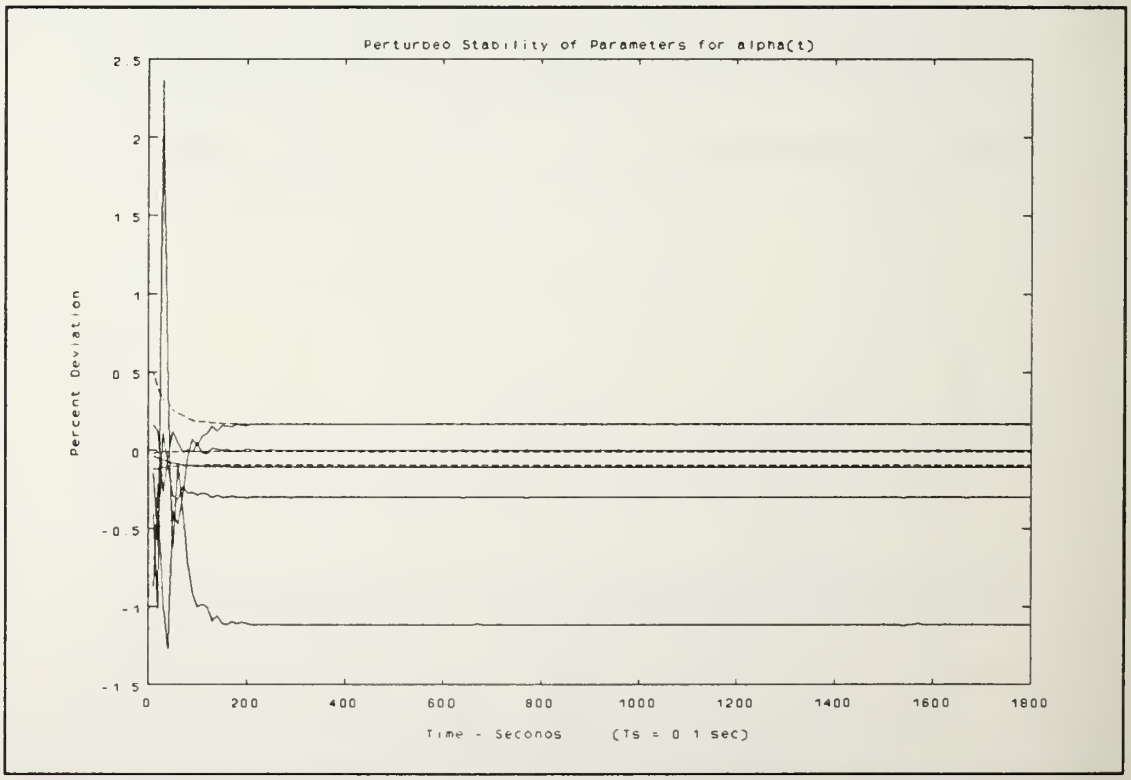

Figure 22: Network Dynamic Stability for $\alpha(t)$

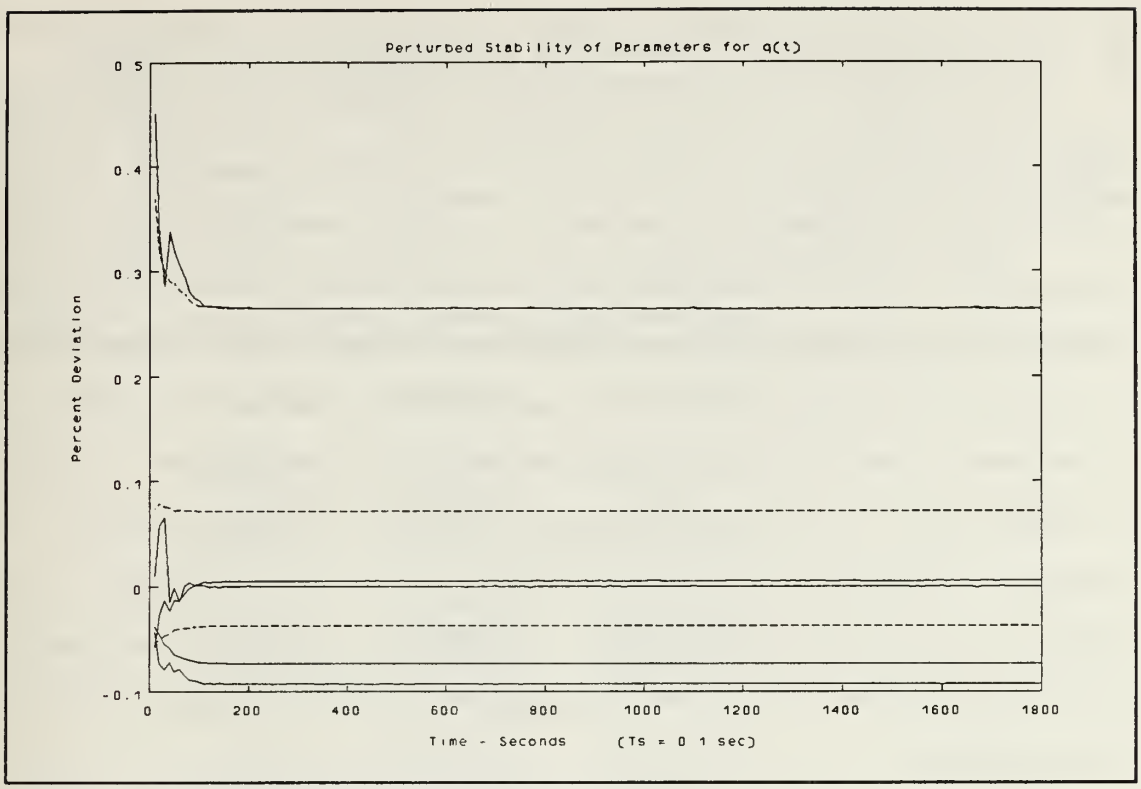

Figure 23: Network Dynamic Stability for q(t)

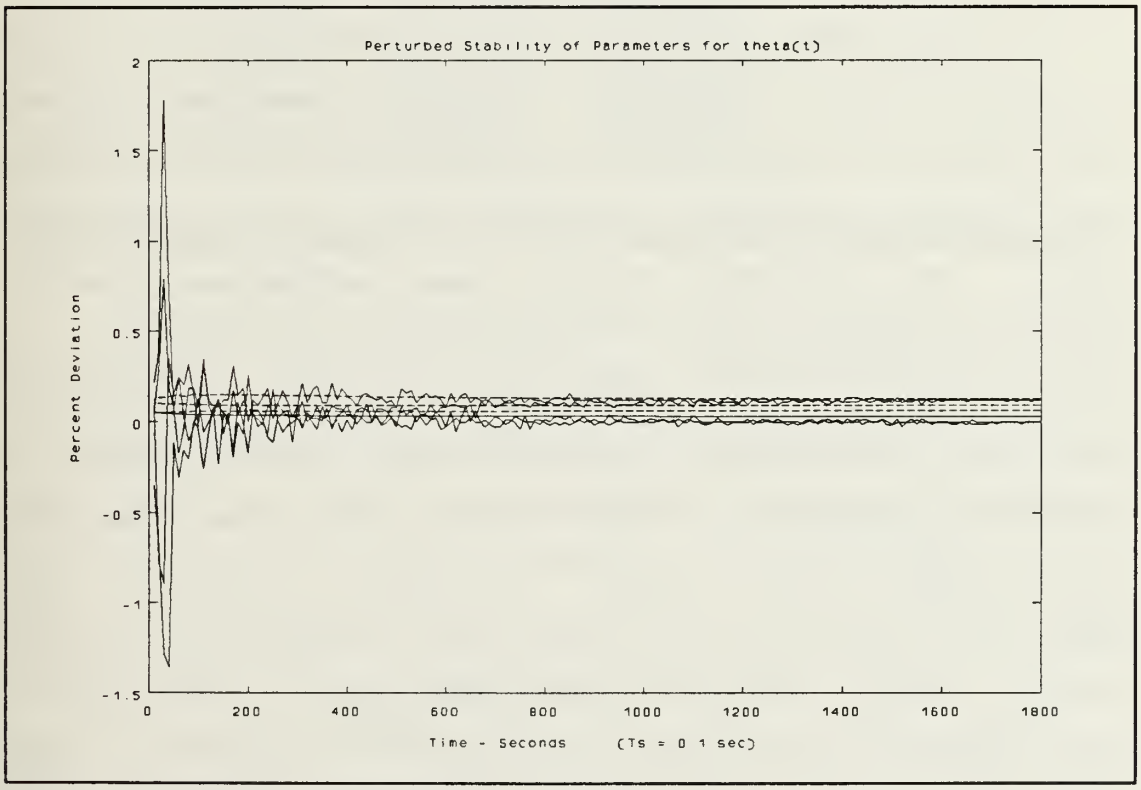

Figure 24: Network Dynamic Stability for Θ(t)

B. THE NEURAL NETWORK ADAPTIVE CONTROLLER IN ESTIMATION

The performance of the neural network adaptive control structure in estimation of the longitudinal motion of the A-4 will be examined in the following paragraphs. Developing a good model of the system to be controlled is important in developing a good control law. The estimation trials were accomplished using the neural network adaptive control structure and skipping the control law synthesis phase of operation for each sample. The performance of two linear networks with different inputs and parameterizations will first be demonstrated in developing a model for the linear system represented by Flight Condition 1 (see Table I.) A nonlinear network will be used to establish a model for the same linear system. The concept of a nonlinear neural network will then be extended to the modelling of multiple flight conditions. The similarities between the performance of linear networks and linear least squares estimators will be shown and extensions provided by nonlinear networks will be demonstrated.

1. Linear Neural Network Parameterized as Four Transfer Functions

Initial estimation efforts involved the use of linear neural networks establishing a model for the condition M 0.4/Sea Level. The first network to be evaluated was trained using the random binary (RB) input and was parameterized as four transfer functions as shown in Figure 15 and discussed in the previous section of this chapter and Chapter V, Section C. This parameterization based on a priori knowledge gives the network 32 parameters to describe the system. It was hoped that the weights of the trained network model would exactly duplicate the parameters used in the simulation. The neural network was trained for 5,000 (500 seconds), 50,000 (5,000 seconds), 500,000 (50,000 seconds), and 5,000,000 (500,000 seconds) cycles. In spite of the lengthy training times, the network never seemed to learn the proper coefficients. The weights for u(t) and $\alpha(t)$, which are representative of the phugoid and short period modes, are compared with the true system parameters in Table IV. The b_i and a_i terms are defined exactly as in equation (5.7). The network determined b_i terms for both u(t) and $\alpha(t)$ appear to be near the correct magnitude

although the signs are incorrect in many cases. No discernible similarities may be seen between the network determined and true values for the a_i terms. What is interesting to note is that the neural network determined coefficients for a_i and b_i are much closer in relative magnitude than those for the true system. The neural network appears to have developed a better balanced or better numerically conditioned representation for the system.

Table IV: Network Weights at 5,000 and 5,000,000 Cycles

Terms	5k Model	5M Model	True Model
b_{u1}	-1.2340e-03	1.6200e-04	2.7131e-05
b_{u2}	-7.4700e-04	1.9200e-04	7.7248e-05
b_{u3}	-3.4000e-04	2.5200e-04	-7.0558e-05
b_{u4}	-8.6500e-04	2.2600e-04	-2.2579e-05
a_{u1}	-7.5005e-02	-1.0065e+00	-3.6949e+00
a_{u2}	-5.9103e-02	-4.8096e-01	5.1802e+00
a_{u3}	-4.3016e-02	1.5285e-02	-3.2755e+00
a_{u4}	-2.6879e-02	4.7618e-01	7.9021e-01
$b_{\alpha 1}$	-3.4585e-02	-3.4585e-02	-3.4619e-02
$b_{\alpha 2}$	-5.9416e-02	-5.9223e-02	4.5022e-02
$b_{\alpha 3}$	-4.2327e-02	-4.2003e-02	1.3715e-02
$b_{\alpha 4}$	-1.2359e-02	-1.2231e-02	-2.4124e-02
$a_{\alpha 1}$	-6.7710e-01	-6.8273e-01	-3.6949e+00
$a_{\alpha 2}$	-4.2595e-01	-4.2207e-01	5.1802e+00
$a_{\alpha 3}$	-6.1324e-02	-5.6638e-02	-3.2755e+00
$a_{\alpha 4}$	4.0465e-01	4.0036e-01	7.9021e-01

The quality of this balanced representation may be better evaluated by conducting time and frequency domain analyses of these models to determine how closely they come to the true system. The swept square wave is a good input to test the time domain response of a model. Figure 25 gives the time and frequency domain characteristics of the swept square wave. The swept square wave is an input signal which excites all of the frequencies of interest. At the same time, the time domain response is easy to visualize since each segment is a unit step input. The plant response of the model trained for 5,000 cycles is shown in Figure 26. The output u(t) exhibits the expected low frequency response, while $\alpha(t)$ and q(t) exhibit high frequency responses, and $\Theta(t)$ exhibits a mix of low and high frequency

responses. These are the expected shapes for the plant response. The RMS prediction error is given in Figure 27. This shows how close the predicted output is to the true output. The network appears to have developed a good model for $\alpha(t)$ and q(t), with RMS errors on the order of 0.1 or ten percent of the maximum output value of one. The network has not, however, learned u(t) and $\Theta(t)$ very well, with RMS errors on the order of 0.7 or almost 70 percent of the maximum output value of one.

Because this is a linear network, the internal structure, the weights, have physical significance. These parameters can be used to evaluate the frequency domain characteristics of the system by generating discrete Bode frequency response plots. The frequency response plots for the longitudinal modes of the A-4 estimated using this parameterization of a linear neural network estimator with a Random Binary Input are given in Figure 28 through Figure 31. In Figure 28, the frequency response for u(t) may be seen. As the training progressed, the network first developed a good high frequency model for u(t), then developed the proper shape for the frequency response, but by 5,000,000 cycles had still not learned the entire response correctly. The presence of unmodelled noise dynamics in the range of frequencies between three and five Hertz is significant. The frequency response for $\alpha(t)$ in Figure 29 shows that the network develops a near exact model almost immediately. However this model does not change much with further training and the network is unable to model the low frequency dipole even after 5,000,000 cycles. The frequency response for q(t) in Figure 30 is similar to that for $\alpha(t)$. In Figure 31, the frequency response for $\Theta(t)$ is similar to that for u(t). Almost immediately the high frequency response (above 0.5 Hertz) is accurately modelled. At 5,000,000 cycles, this accurate modelling has only expanded down to about 0.1 Hertz. Again, there is some undesirable high frequency noise modelling present. As expected from the discussion of input selection in Chapter V, the frequency response for all outputs is good over a limited range of frequencies, even though the random binary input is known to be persistently exciting. At the same time, some outputs show the modelling of undesirable noise dynamics in the very high frequencies related to the concept of information content. The low frequency dynamics have apparently been lost and replaced by some high

frequency noise dynamics. The neural network is more sensitive to high frequency noise dynamics excited by the persistently exciting random binary input than to the low frequency system dynamics. The concepts of persistency of excitation, information content, and the effects of sampling time can all be seen in the frequency response results for this network parameterization. The network (see Table IV) appears to be making the best balanced realization it can with the available parameters. From these results it may be seen that neural network estimators are governed by some of the same precepts that govern traditional estimation.

Various other trials were conducted using a network parameterized as four transfer functions with little improvement on the results. Inputs with excitation in different frequencies, slightly overparameterized systems, and different sampling times were used to attempt to obtain better results. The network parameterized in this manner could not learn both the high and low frequency dynamics at the same time.

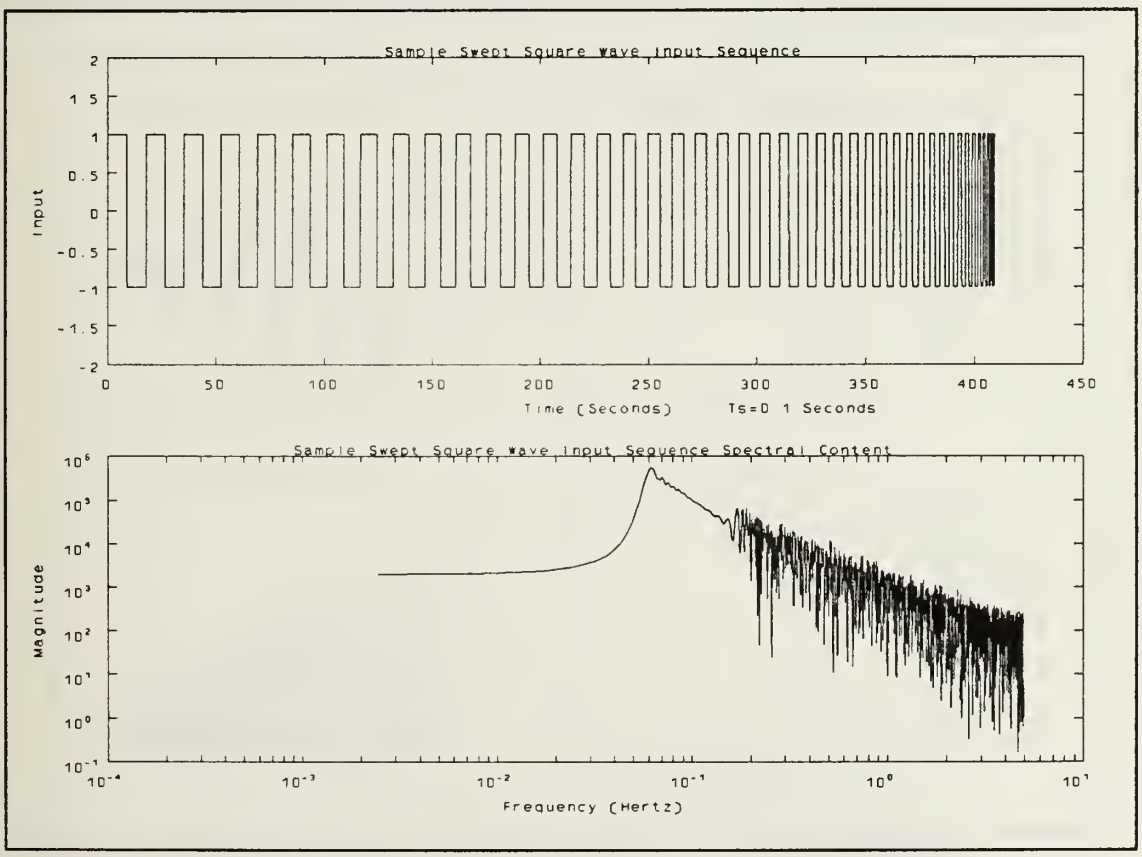

Figure 25: Input Characteristics

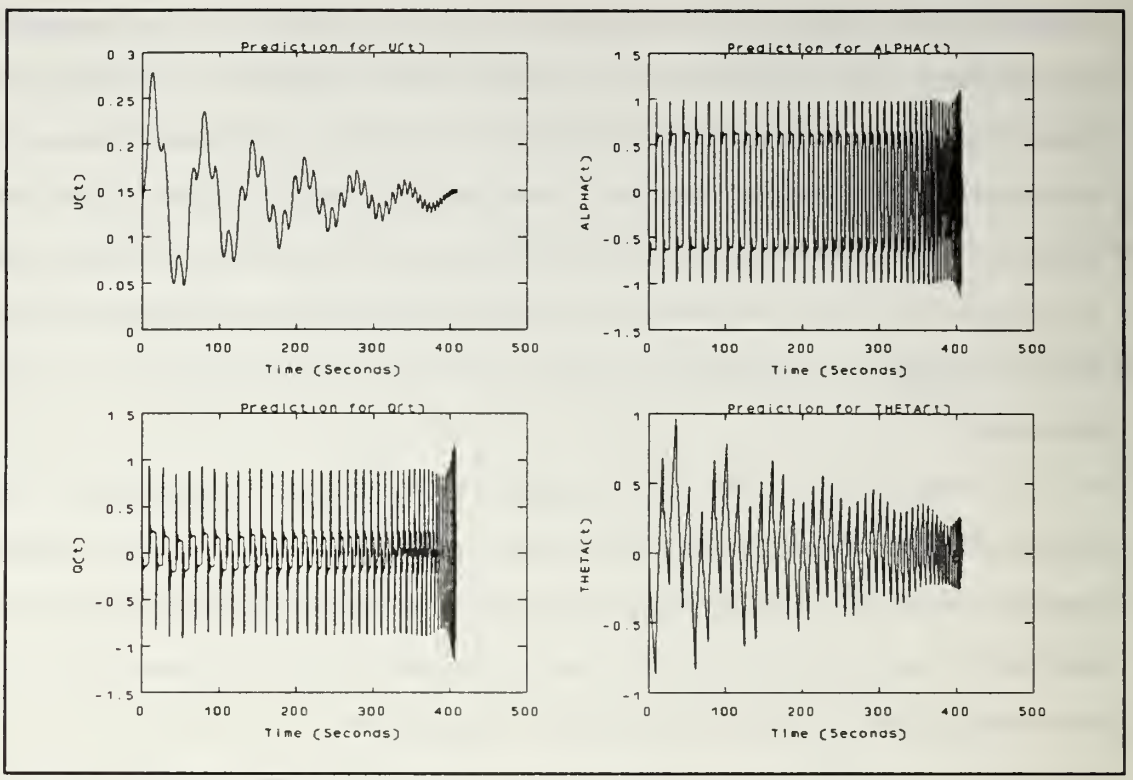

Figure 26: Plant Response

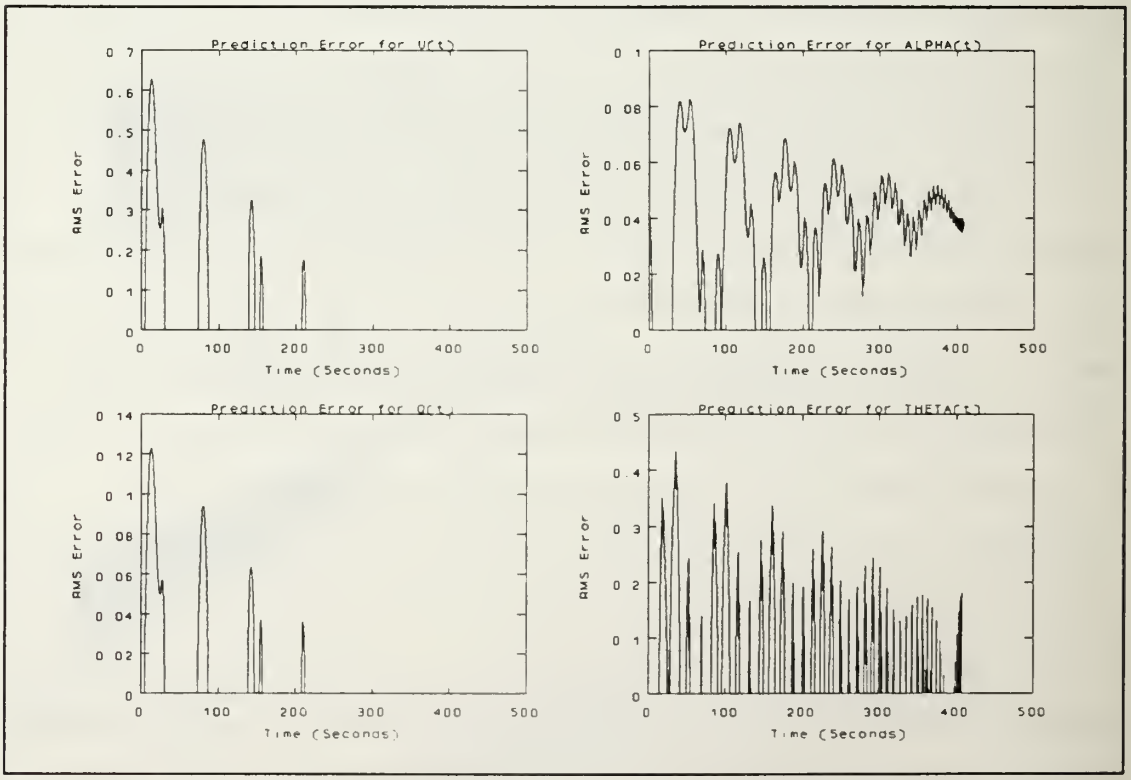

Figure 27: Prediction Error

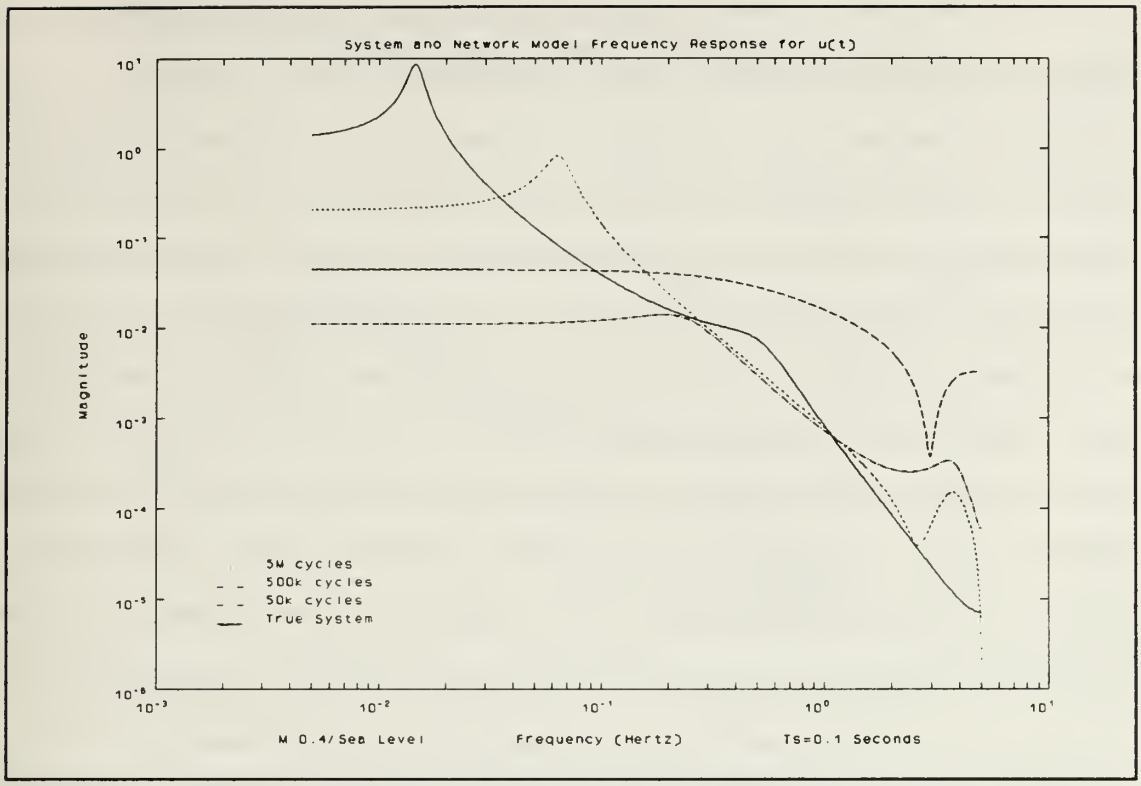

Figure 28: Frequency Response for u(t) for Various Amounts of Training

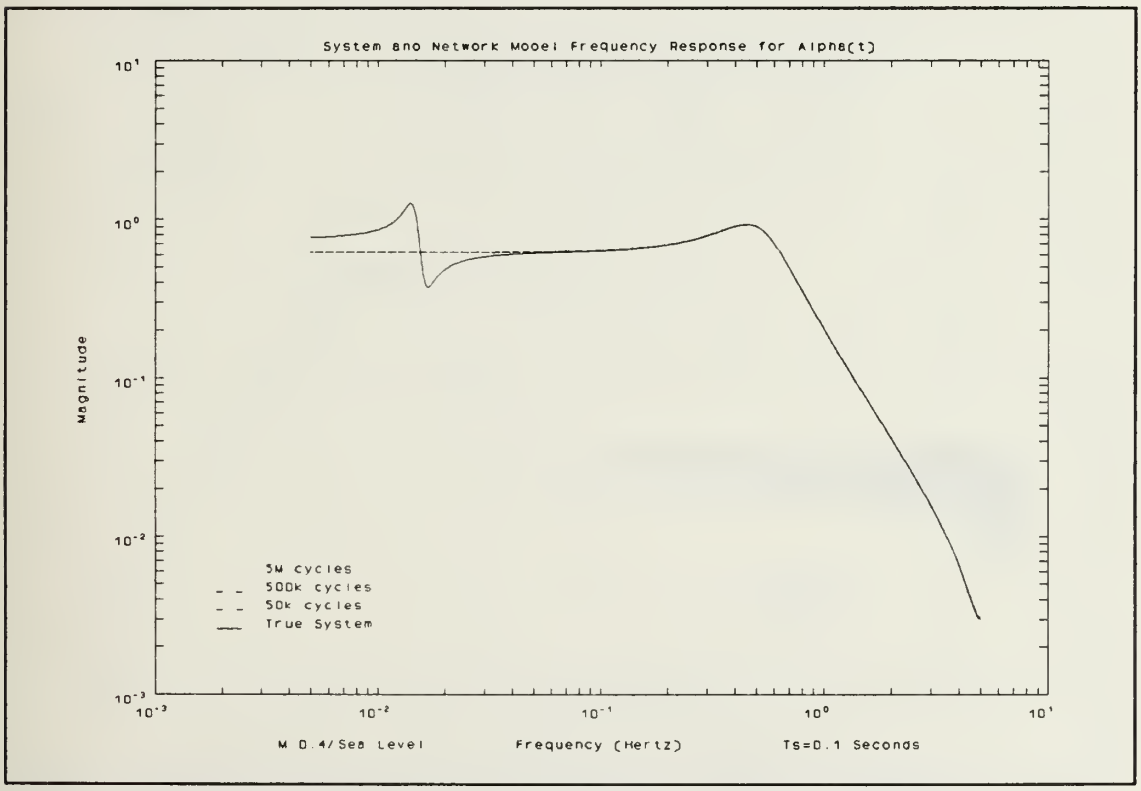

Figure 29: Frequency Response for $\alpha(t)$ for Various Amounts of Training

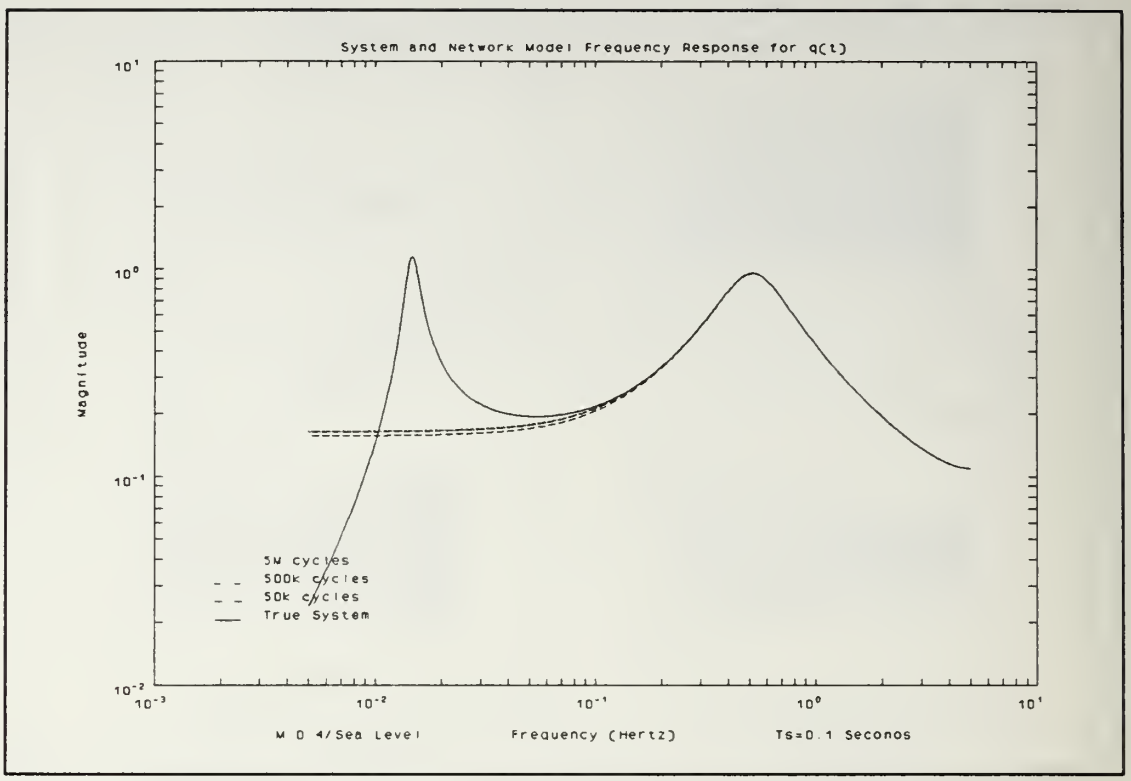

Figure 30: Frequency Response for q(t) for Various Amounts of Training

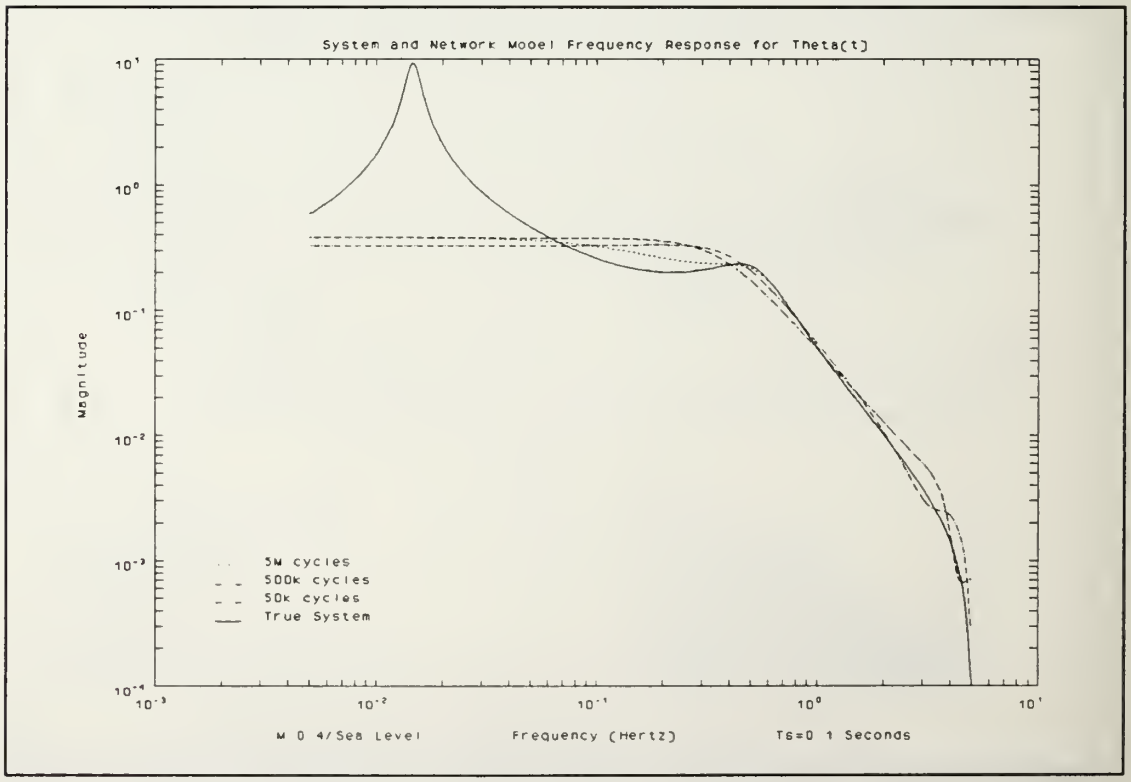

Figure 31: Frequency Response for Θ(t) for Various Amounts of Training

2. Fully Connected Linear Neural Network

The second linear neural network model to be demonstrated is fully connected, i.e. it has all of the input elements connected to each of the output elements. The belief was that the network was not being given enough parameters to describe the system, including any noise dynamics. This highly overparameterized neural network has a parameterization similar to that suggested by [Ref. 6:pp. 115-126] for multivariable systems. It is interesting to note that the fully connected neural network, which is intuitively a more natural parameterization, is similar to that recommended for multivariable systems. The 32 parameters used in the previous example are replaced by 80 parameters. The input was also slightly modified. The random binary (RB) input was bandlimited by allowing it to change every two samples instead of every sample. The resulting pseudo random binary (PRB) input is shown in Figure 32. The severe drop in spectral energy above four Hertz was intended to

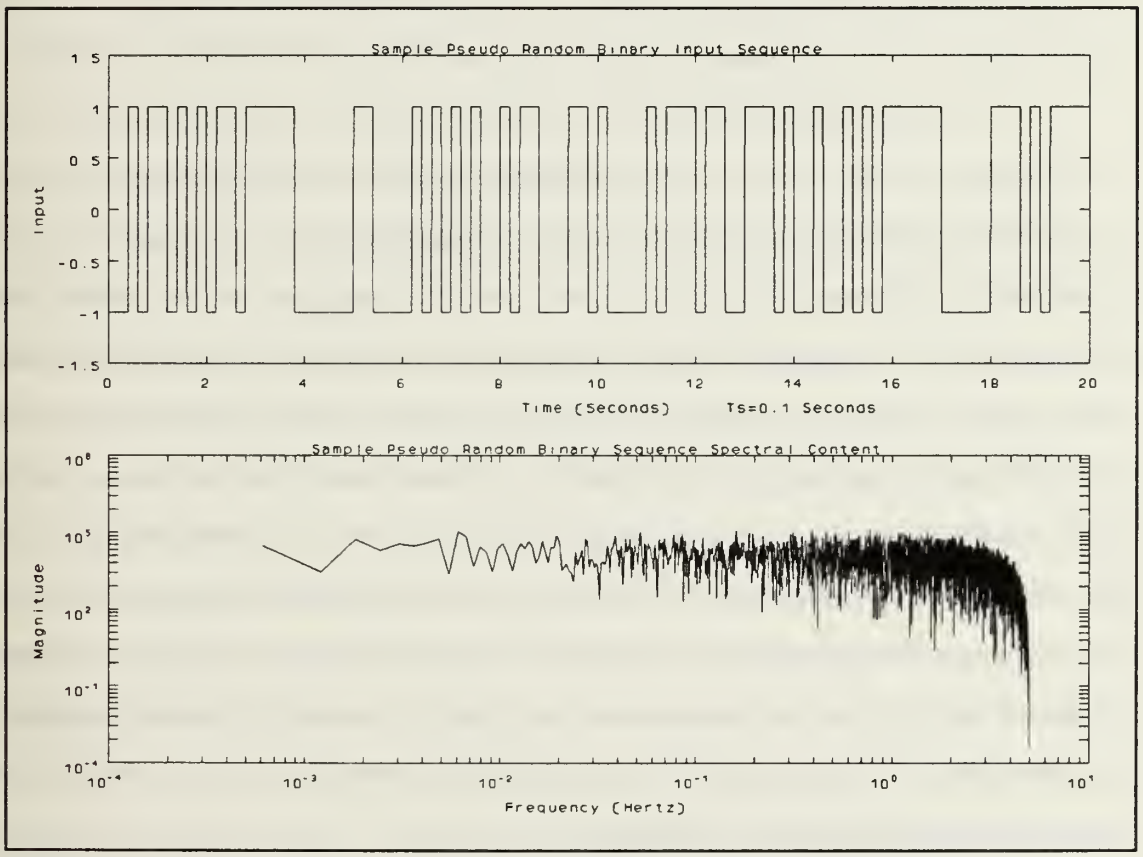

Figure 32: Pseudo Random Binary Input Sequence and Spectral Content

limit the excitation of the high frequency noise. Using a network parameterized in this fashion, the dynamic estimation error for all outputs went to zero in less than 5000 cycles (500 seconds). The error for each output resulting from testing this model with the swept square wave in Figure 25 was smaller than the precision of the Neuralworks program as shown in Figure 33. Comparison of the specific parameters in this model with those for the true system is difficult. This network appears to have developed a near exact model for the input-output relationship of the true system. Two factors allow this network to perform far better than the network parameterized as four transfer functions. The highly overparameterized nature of this network allows parameters to be used to model noise dynamics and provide a better balanced representation of the system. At the same time, the fully connected structure allows crosstalk between outputs, providing a means for outputs to develop dependencies on past values of other outputs. Thus, through the use of better parameterization and a bandlimited input, the linear neural network performed very well at the estimation of the longitudinal motion of the A-4 aircraft.

A linear neural network can successfully produce a near exact model for the A-4 longitudinal modes. However, neural network estimators are limited by the same concepts of persistent excitation, information content, and sampling time as least squares estimators introduced in Chapter III. From the first example, it appears that the neural network attempts to make the best balanced realization of the model possible with the given number of parameters. From the second example, the use of a fully connected neural network estimator proved to be much more successful. This result lends some credence to the use of fully connected neural networks and demonstrates the ease with which neural network parameterizations can be changed. By demonstrating two examples of a linear neural network in parameter estimation, the natural manner in which estimation problems can be represented in neural network structures has been shown, as well as the similar effects of concepts such as persistency of excitation, information content, parameterization, and sampling time in neural networks and classical estimators.

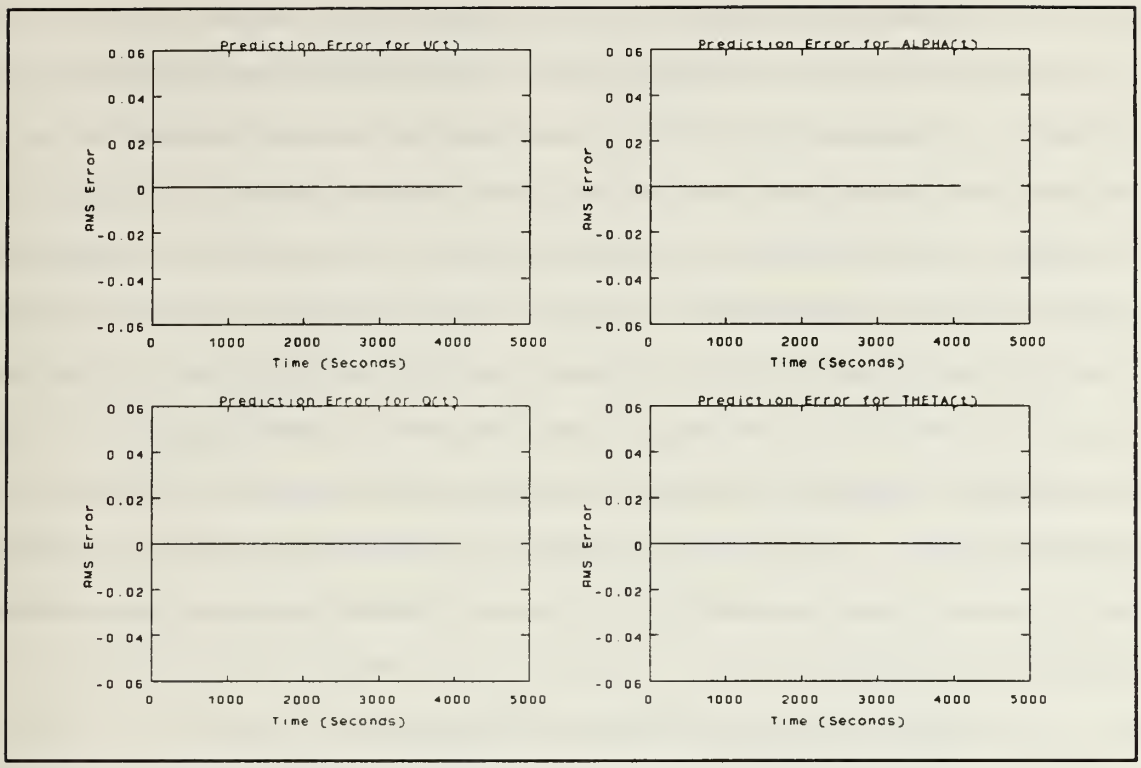

Figure 33: Prediction Error for Fully Connected Network

3. Nonlinear Neural Network Estimators

The power of Backpropagation lies not in its ability to model linear systems, but in its ability to model nonlinear systems. In the following paragraphs, the use of this capability to model systems will be investigated. The use of a nonlinear network to develop a model for the linear system represented by Condition 1 will be discussed. This nonlinear neural network will then be extended to the modelling of the nonlinear system represented by multiple flight conditions. The estimation process in the neural network adaptive control structure is made nonlinear by the addition of one or more hidden layers of elements with nonlinear activation functions inside the network internal system model as discussed in Chapter V, Section C. Nonlinear neural network estimators pose two problems in addition to those described in Chapter V for all estimators. First is the selection of the number of elements in the hidden layer. This is done empirically due to the lack of any other method. For the single condition network, the same number (20) of elements were used in the hidden

layer as the number of elements in the regression vector. For the multiple condition network, 160 elements were used in the hidden layer. The other problem involves determination of the gain used in equation (2.12). For the linear neural network, this gain could be absorbed directly in the learning rate, however for nonlinear activation functions, the gain is inside the function. The most important effect of this gain is on the sensitivity of the activation function to inputs. For semilinear activation functions, a high gain causes the activation function to approximate the signum function, taking on values of -1.0 and +1.0 for almost all values of the input. A very low gain causes the activation function to behave in a more linear fashion. Again, determination of the activation function gain is done empirically. For this investigation, values for the gain between 0.5 and 1.0 were used. The use of nonlinear neural networks in estimation adds considerable capability at the expense of some additional complexity.

4. Nonlinear Network Modelling Linear System

The first nonlinear neural network estimator was trained using the M 0.4/Sea Level condition. The layers were fully connected. The input used was the original random binary input. The order of the dynamic RMS estimation error at 5,000 cycles (500 seconds) was on the order of 0.05, or five percent of the maximum value, for each output compared with nearly zero for the fully parameterized linear network described above. By 50,000 cycles (5,000 seconds) the dynamic RMS estimation error was on the order of 0.01 for each output. Providing ten times the training did not significantly change the amount of error in the system. This represents better performance than that for the linear neural network estimator parameterized as four separate transfer functions, but worse than the performance of the fully connected linear neural network estimator. A frequency domain analysis of this model may help to better understand the performance of this network. It is impossible to produce typical frequency response plots for nonlinear systems, however, spectral transfer functions can be developed from the input-output data. This is done by recording model input and output sequences. The sequences are then windowed and transformed into the complex frequency domain using a fast fourier transform. A complex transfer function is

developed by dividing the complex output spectrum by the complex input spectrum at each frequency. The magnitude and phase characteristics of the transfer function may be approximated by the magnitude and phase of this spectral transfer function. In this case, the input and output sequences were 9000 cycles long, and the data was broken up into overlapping 2048 point segments which were windowed using the Hanning method. The resulting spectra are averaged to smooth out the curves. This is known as Welch's method. Further information on this technique is available in [Ref. 11]. The spectral transfer functions for the 5,000 and 50,000 cycle models are given in Figure 34 through Figure 37. Figure 34 shows the spectral transfer functions for u(t) where the frequency response is well modelled across the spectrum with the exception of small errors in the very low frequencies and considerable noise in the higher frequencies. Note also that increases in training do little to improve the model. The same is true for the spectral transfer functions for $\alpha(t)$ shown in Figure 35. The spectral transfer functions for q(t) in Figure 36 are very close to the true system frequency response and there is little high frequency noise. The spectral transfer functions for $\Theta(t)$ in Figure 37 exhibit the same model and noise characteristics as u(t) and $\alpha(t)$. Note that the models are all relatively good, however, the very low and very frequencies are corrupted. Developing better nonlinear models is a topic which deserves further study.

5. Multiple Condition Nonlinear Neural Network Estimator

The nonlinear neural network estimator described above may be easily extended to the modelling of nonlinear systems. This is done by incorporating some measure of the nonlinearity in the regression vector. For this investigation, the nonlinearity is provided by including the mach number and altitude in the regression vector as discussed in Chapter V. Four of the conditions described in Chapter V (Conditions 1, 3, 4, and 5) were used to train the network which was tested on the fifth (Condition 2). The network was trained for 36,000 cycles (3600 seconds) and 360,000 cycles (36,000 seconds) with the condition changing every 9000 cycles. The results for conditions on which the network was trained were similar to those already described in Figure 34 through Figure 37. The same type of results for the

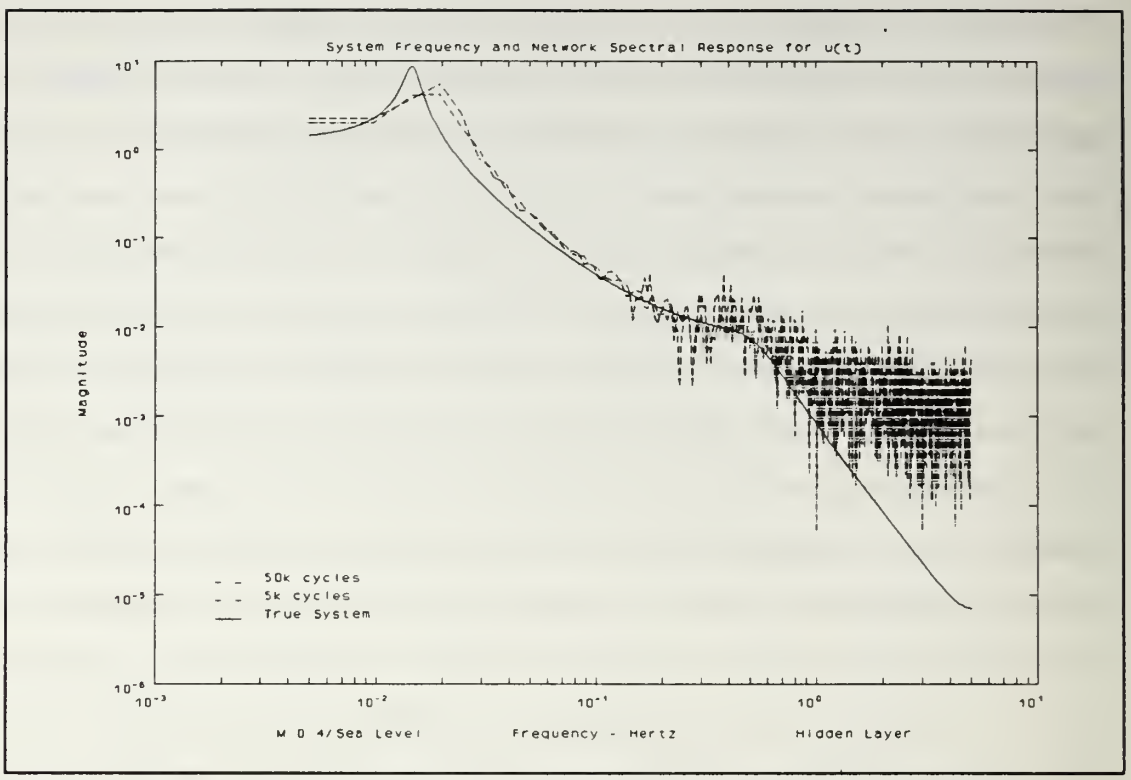

Figure 34: Spectral Transfer Function for u(t)/Nonlinear Hidden Layer

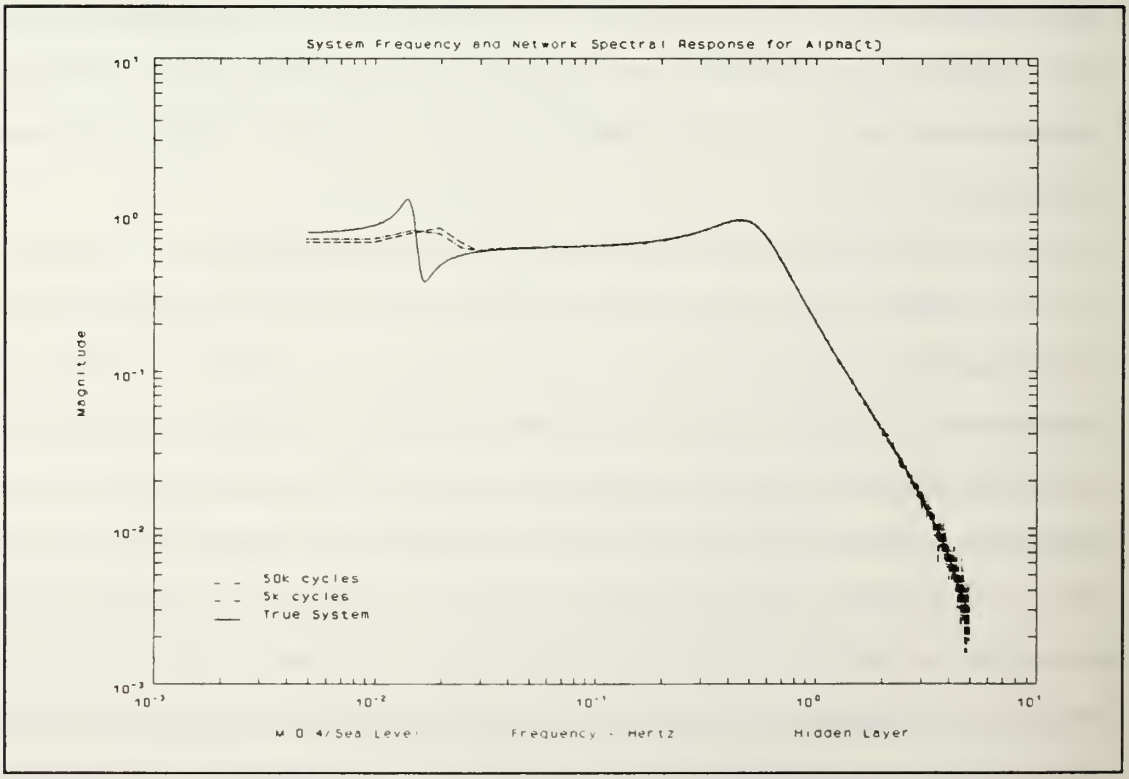

Figure 35: Spectral Transfer Function for α(t)/Nonlinear Hidden Layer

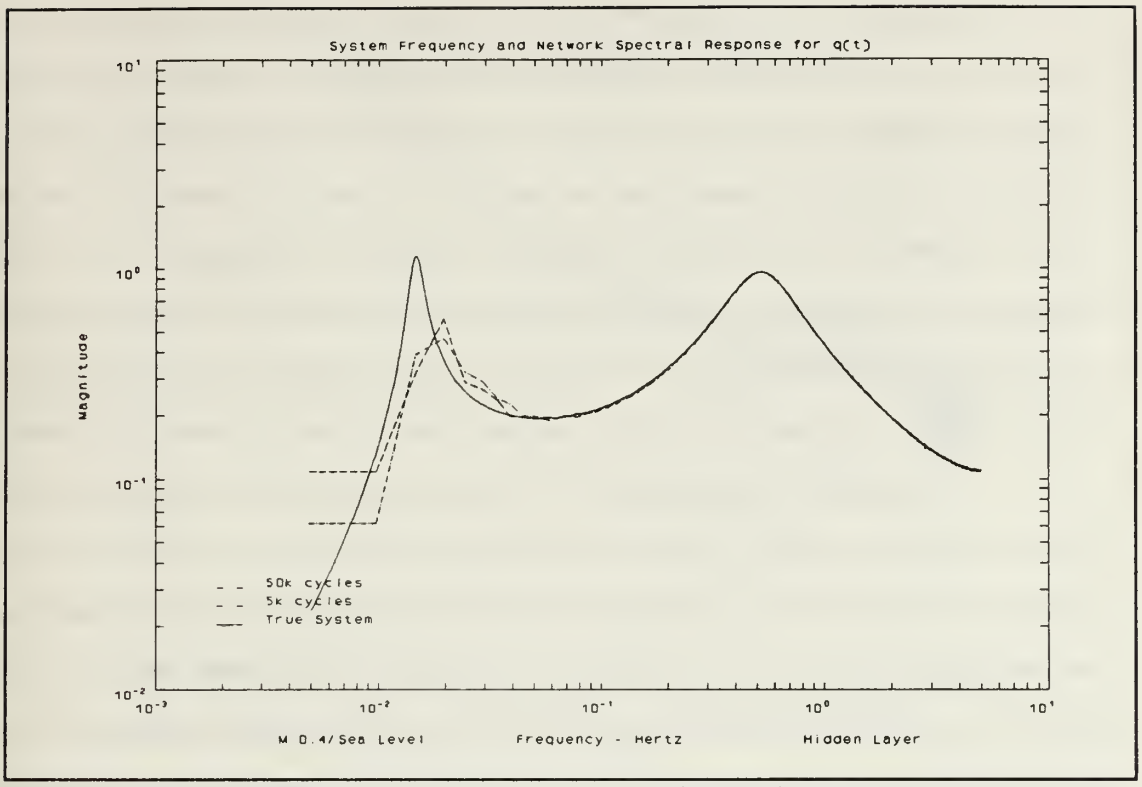

Figure 36: Spectral Transfer Function for q(t)/Nonlinear Hidden Layer

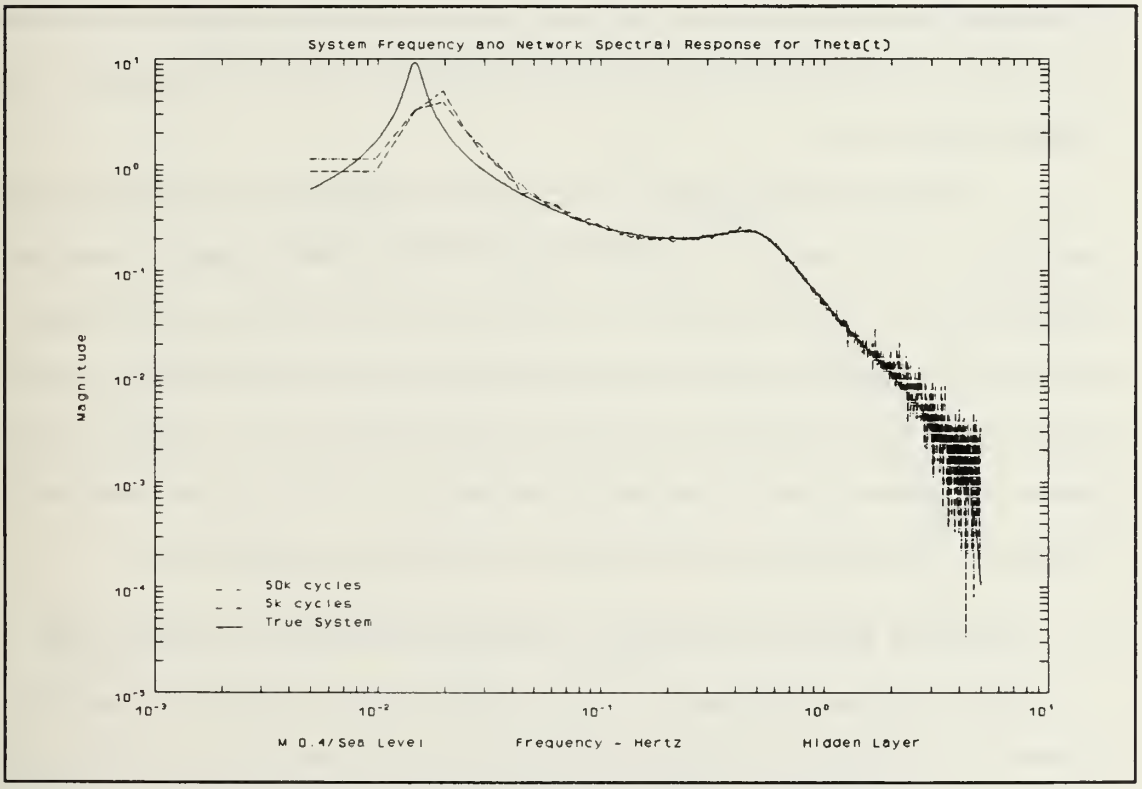

Figure 37: Spectral Transfer Function for Θ(t)/Nonlinear Hidden Layer

untrained condition (M 0.5/15,000) are shown in Figure 38 through Figure 41. The spectral transfer function for u(t) for the untrained condition is shown in Figure 38. The frequency response is relatively good with the exception of significant noise in the frequencies above 0.5 Hertz. The spectral transfer function for $\alpha(t)$ in Figure 39 is much better across the entire frequency range, however some noise dynamics are apparent in the range of frequencies above one Hertz. The spectral transfer functions for q(t) in Figure 40 show good response across the entire spectrum. The spectral transfer functions for $\Theta(t)$ in Figure 41 are similar to those for $\alpha(t)$ and u(t) with some noise dynamics present in the higher frequencies.

An interesting phenomena is the fact that at the moment the simulation changes from one flight condition to another, the dynamic RMS error rises sharply then drops, indicating that the network is 'relearning' that particular condition. This may imply that the network model selection is not sufficient to fully model the nonlinearities of the multiple models. This method provides a good model for trained and untrained conditions. However, further study is again warranted to determine whether a better model structure may be found to represent the nonlinear system. Nonlinear neural networks can be effective in modelling nonlinear systems, however some further work on determination of the number of hidden elements must be done.

The use of neural networks in estimation has been demonstrated in the previous paragraphs. The similarity between neural networks and classical estimators was demonstrated, as well as the ease with which neural networks can be reconfigured. The power of the network to choose its own parameterization was also demonstrated. Justification for the use of fully connected neural networks was described. Also, the use of nonlinear activation functions to model nonlinear systems was shown. The structure developed for the neural network adaptive controller is successful at estimation applications.

C. CONTROL USING THE NEURAL NETWORK ADAPTIVE CONTROLLER

The neural network adaptive control structure is also a successful controller. An important issue in the use of adaptive controllers is the fact that closed loop systems corrupt

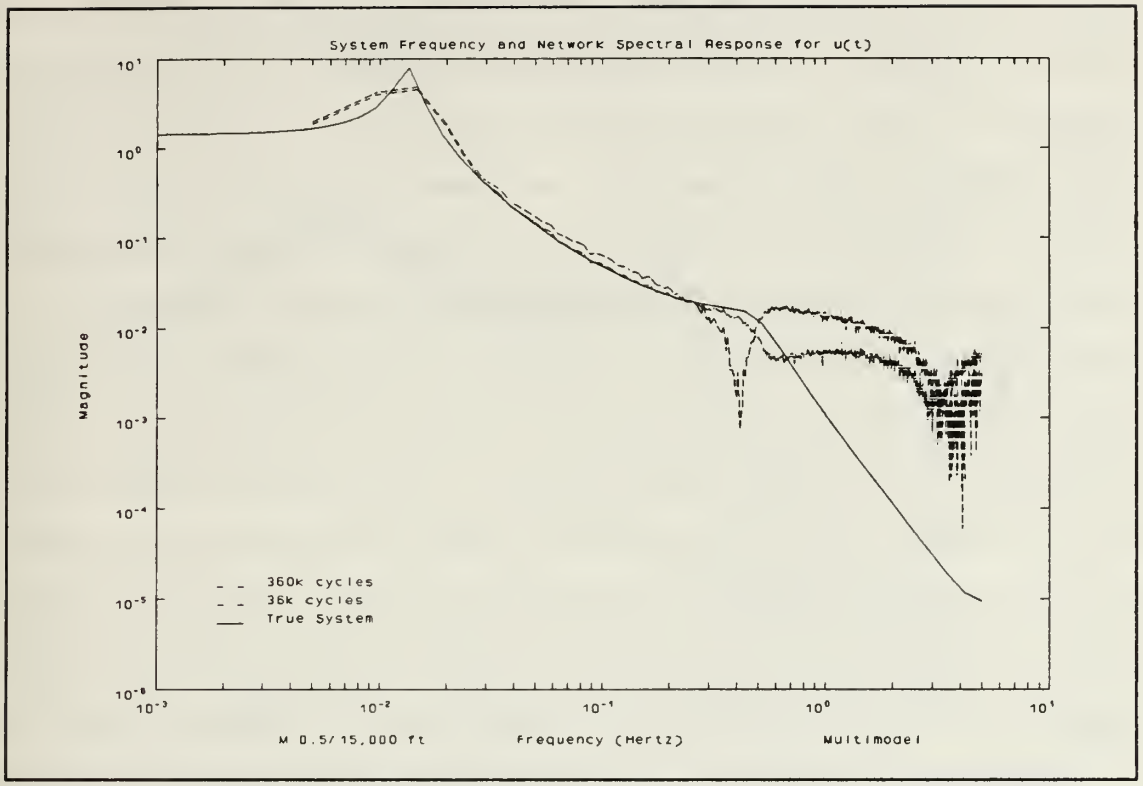

Figure 38: Spectral Transfer Function for u(t)/Untrained Condition

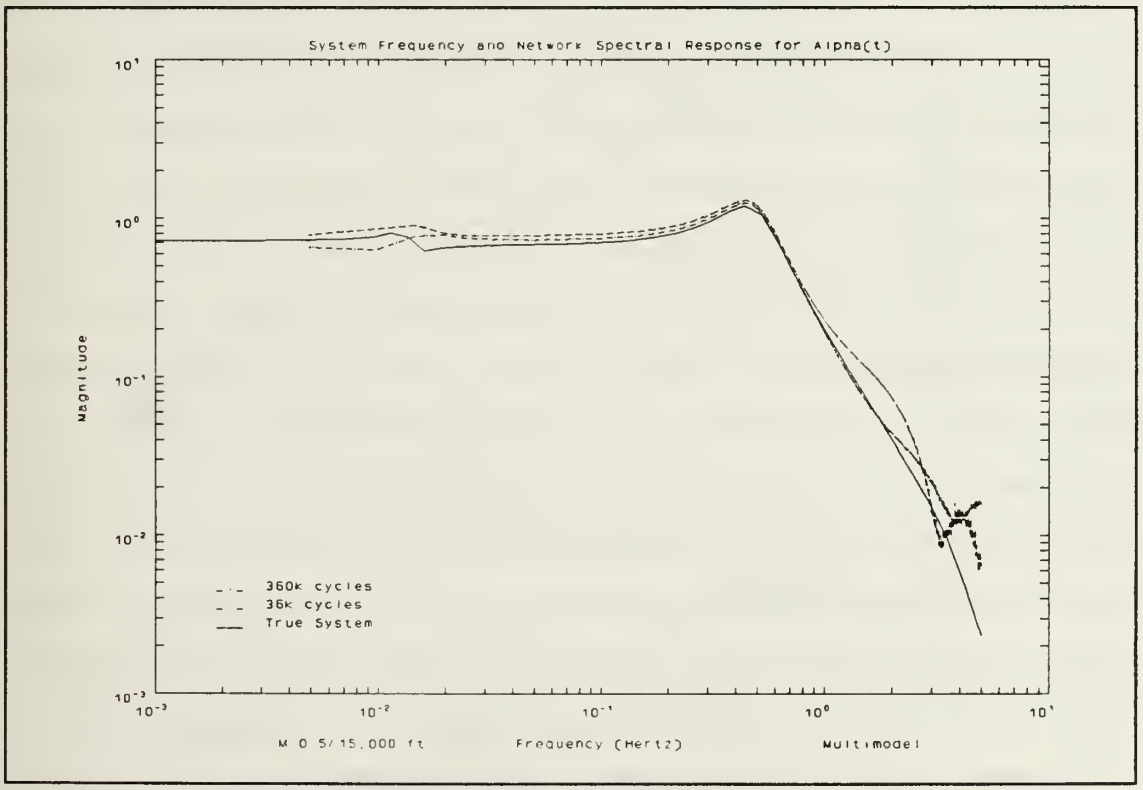

Figure 39: Spectral Transfer Function for α(t)/Untrained Condition

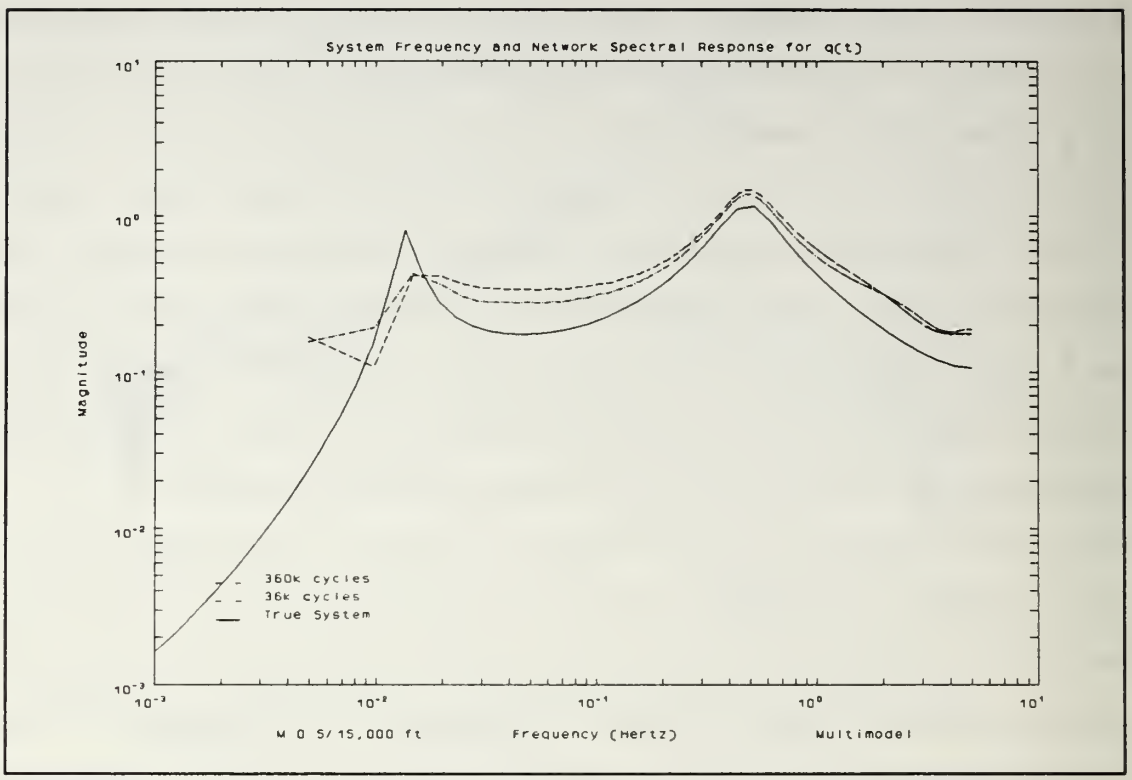

Figure 40: Spectral Transfer Function for q(t)/Untrained Condition

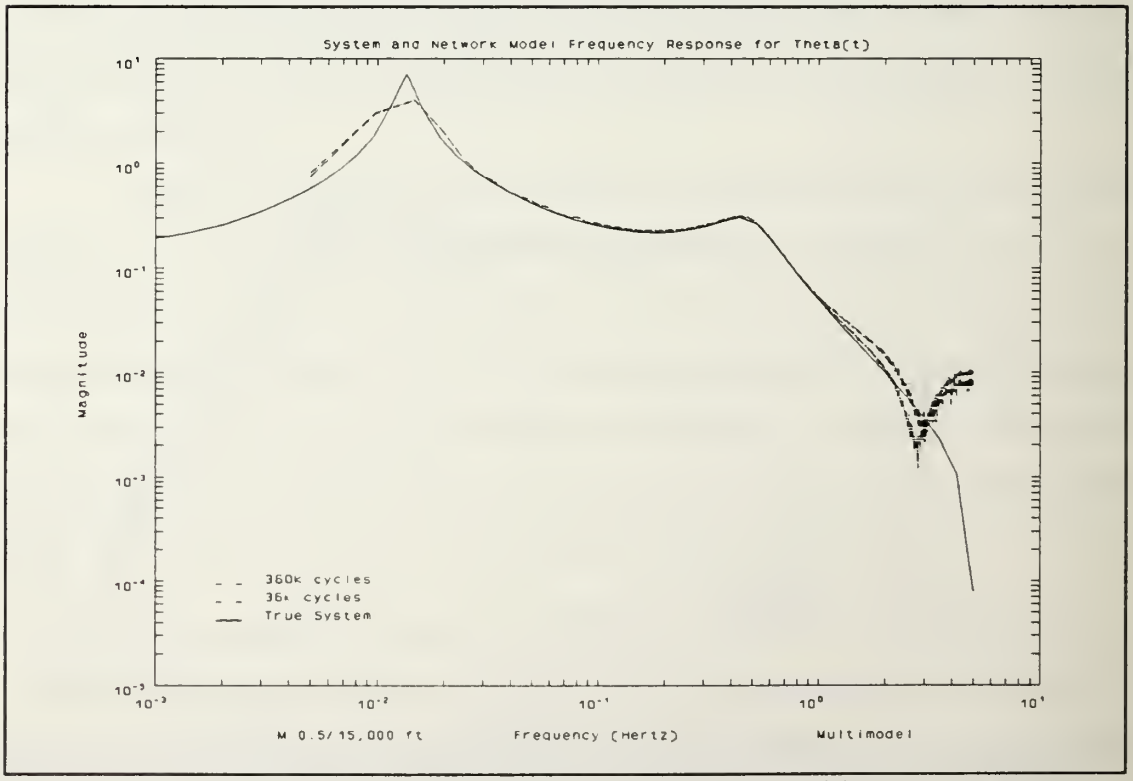

Figure 41: Spectral Transfer Function for Θ(t)/Untrained Condition

the input. This may eliminate any persistency of excitation present in the input. The model does not have to be exact, however, to produce a good controller. The model only has to be accurate over the range of frequencies in which the controller is active. This should be provided by the control inputs themselves. The simple example of a neural network demonstrated here shows the noise rejection capabilities of a linear neural network controller.

Using the random binary input, the noise rejection capabilities of the neural network adaptive control structure can be demonstrated. Since adaptive controllers have proven effective in noise rejection in the past, it was hoped that the neural network adaptive controller would be effective as well. The network used for this part of the investigation was the same as that used for the fully parameterized linear neural network estimator. The flight condition was Condition 2 with a mach number of 0.4 and an altitude of Sea Level. The plot in Figure 42 shows the control input which the network produced in response to the random binary input for the first 9000 cycles (900 seconds). Note that the input goes asymptotically to zero as the network rejects the white noise, random binary input. The network dynamic RMS estimation and tracking errors are shown in Figure 43 through Figure 46. The dynamic estimation error is the error in the model or prediction as the network trains. The tracking error is the difference between the model reference output and the network output as the network trains. In Figure 43 the tracking error and estimation error both appear to go to some small value. The estimation error and tracking error for $\alpha(t)$ in Figure 44 are much more descriptive. Note how the estimation error goes to zero while the tracking error goes to some steady state value. The same result can be seen in the estimation and tracking error for $q(t)$ in Figure 45 and for $\Theta(t)$ in Figure 46. The estimation error for all of the outputs is decreasing, but has not gone to zero as in the fully parameterized linear neural network estimator. This is due to the loss of persistent excitation in the control input. Also, note the fact that the tracking error for all four outputs goes to some steady state error value. Within the 9000 cycles (900 seconds) that this model was trained, the white noise random binary input signal was rejected. In this simple example, the use of a neural network adaptive controller to reject noise has been effectively demonstrated.

D. SUMMARY

The effectiveness of neural network adaptive controllers in estimation and control has been demonstrated in this chapter. The static and dynamic stability of the neural network adaptive control structure was shown. The effects of persistency of excitation, information content, and sampling time on the estimation process were demonstrated using a linear network. With the addition of more parameters, this linear network could develop an exact model for the system. The use of nonlinear neural networks in estimation was then demonstrated to develop models for linear and nonlinear systems. The importance of developing the theory necessary to use nonlinear neural networks was discussed. Finally, the use of a simple neural network adaptive controller in noise rejection was demonstrated and the effects of adaptive control on estimation were discussed. This neural network adaptive control structure shows tremendous promise for future applications.

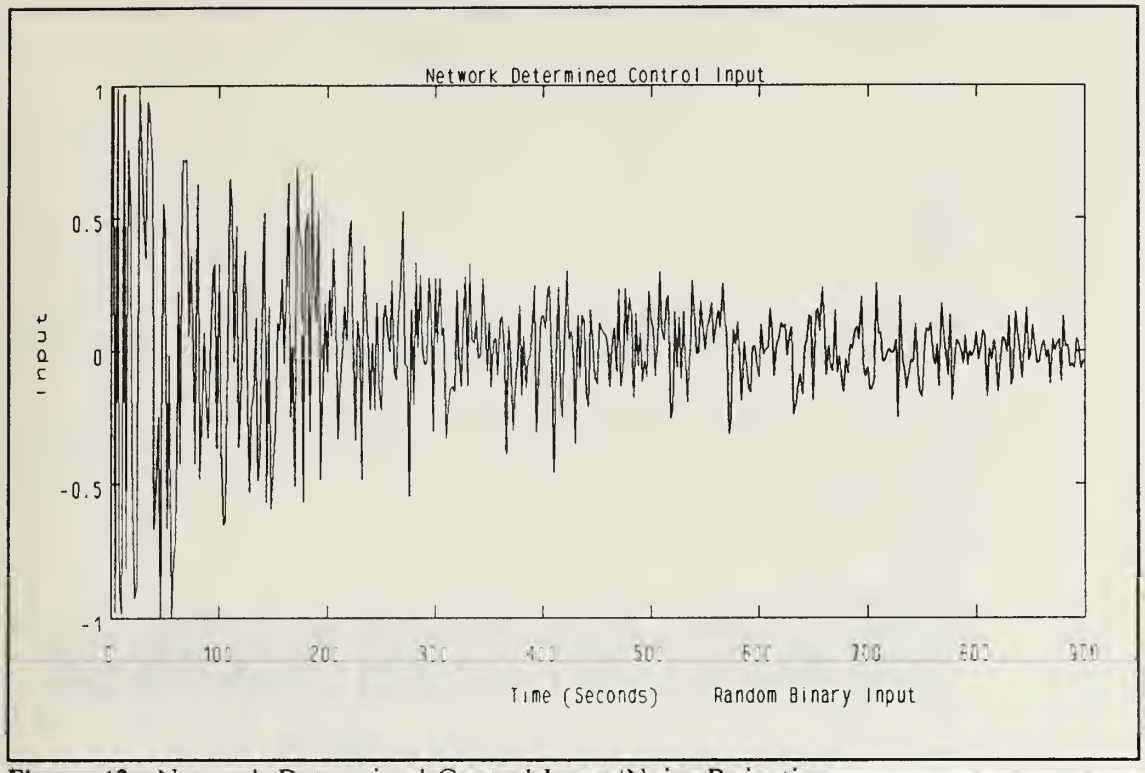

Figure 42: Network Determined Control Input/Noise Rejection

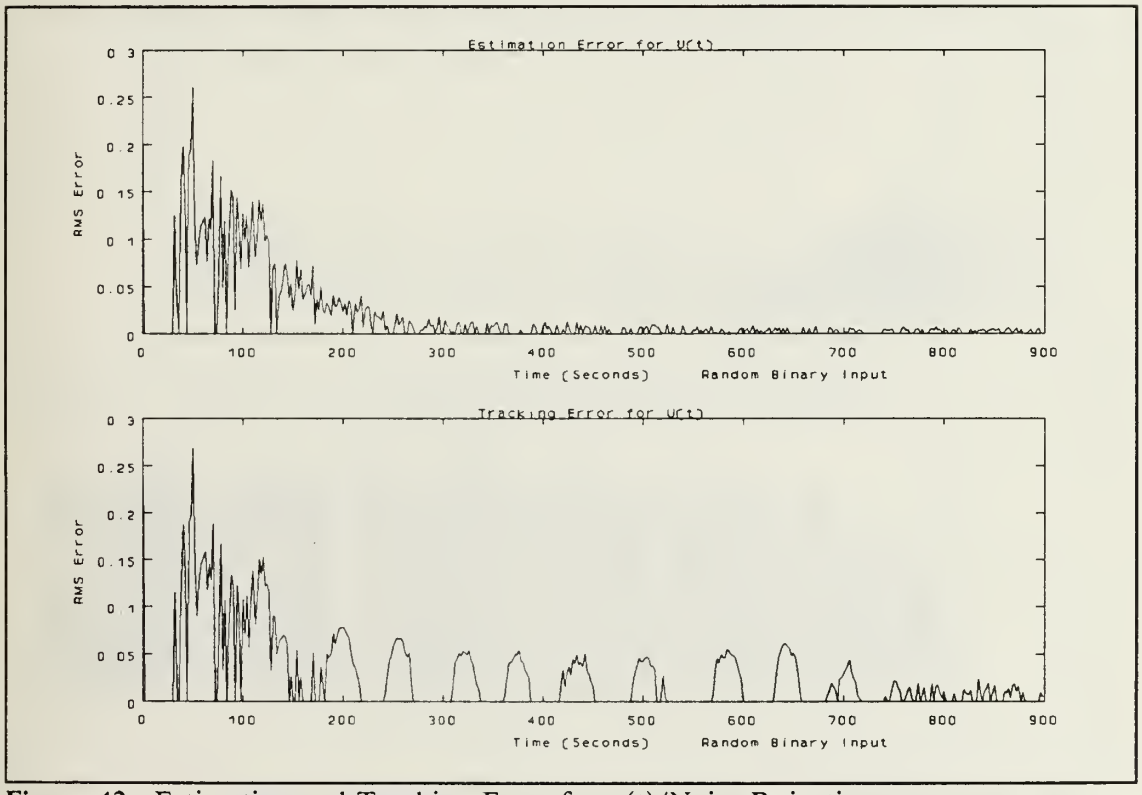

Figure 43: Estimation and Tracking Error for u(t)/Noise Rejection

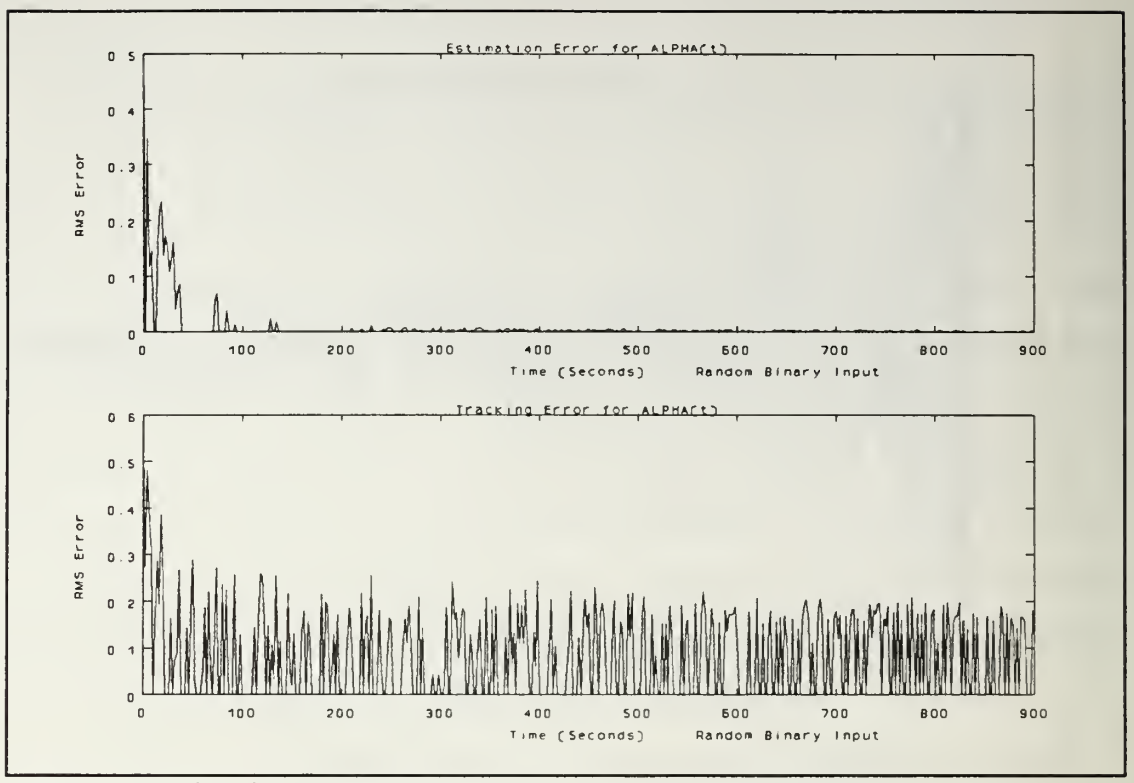

Figure 44: Estimation and Tracking Error for $\alpha(t)$/Noise Rejection

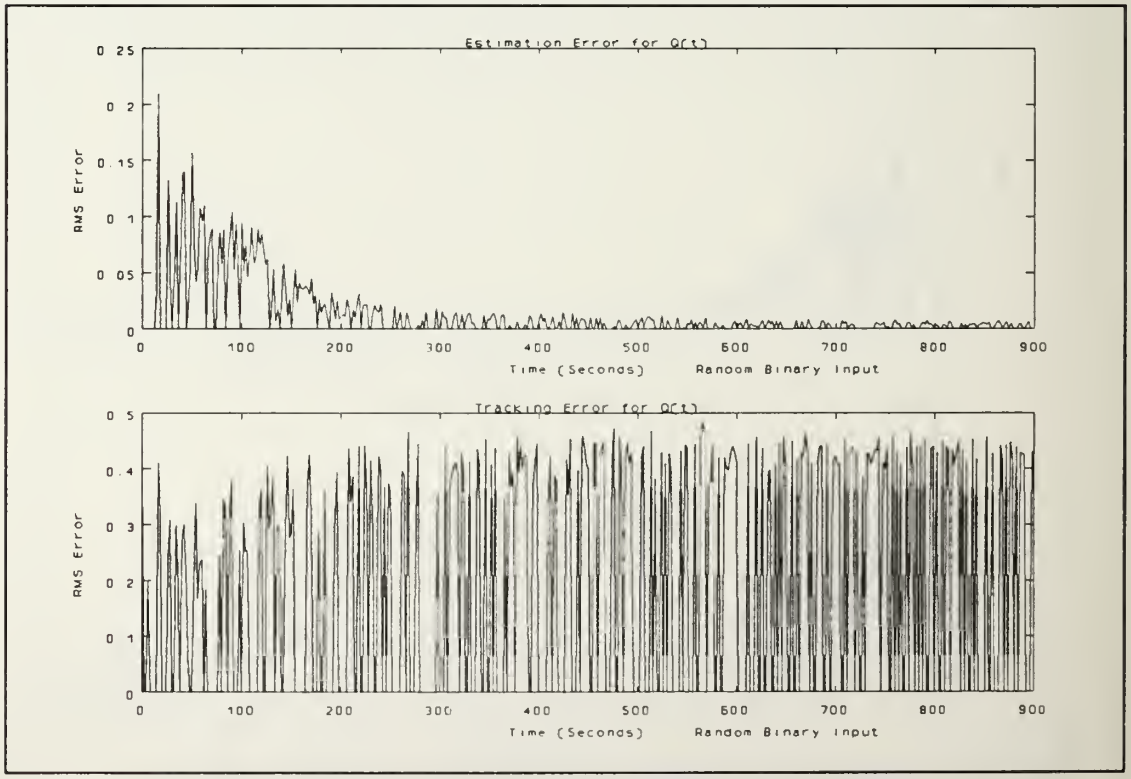

Figure 45: Estimation and TRacking Error for q(t)/Noise Rejection

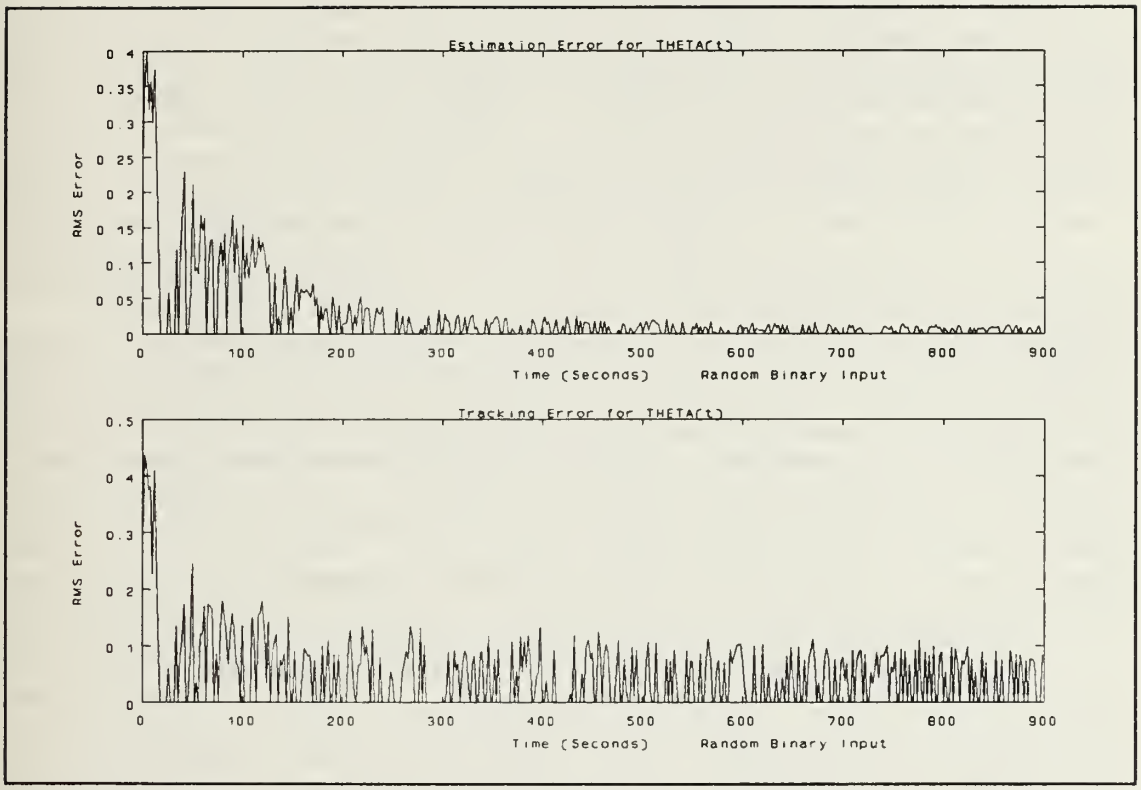

Figure 46: Estimation and Tracking Error for Θ(t)/Noise Rejection

VII. CONCLUSIONS AND RECOMMENDATIONS

Neural networks are effective in the solution of adaptive control problems. As systems become more complex and the requirements placed on them become more demanding, parallel distributed processing applications will become an important tool in the design of adaptive controllers. In this thesis, a neural network adaptive control structure was developed from similarities in neural network, estimation, and control theory. The effectiveness of this structure was tested in the estimation and control of linear and nonlinear approximations of the longitudinal motion of the A-4 aircraft. The difficulties which the system of longitudinal motion of the A-4 aircraft presents to conventional estimation and control were discussed. Various significant concepts in estimation and control were discussed and demonstrated using the neural network adaptive control structure. The significance of parameterization for estimation applications as well as neural networks in general was illustrated. A theoretical basis for the scaling of data and the choice of learning rate in neural networks was developed. The concept of the semilinear activations providing robust linear characteristics was discussed. The neural network adaptive control structure developed for this thesis demonstrated the applicability of parallel distributed processing tools to adaptive control.

The neural network adaptive control structure introduced in this thesis was developed in a manner consistent with adaptive control theory. The concept of estimation involving the mapping of some regression vector of past input and output measurements into the current output measurement was developed. The idea of a general control structure involving the weighted sum of some state variable and a reference input was also discussed. From these concepts, a structure for a neural network adaptive controller involving an estimation and control process using the Backpropagation neural network type was determined. As an estimator, the neural network maps a regression vector into a current measurement. As a controller, the neural network maps the regression vector into a control input, which is then fed forward through an internal model of the system and compared to some reference output

in order to adjust the weights, or gains, of the controller. Unlike conventional adaptive control schemes, the neural network adaptive controller is easily extended to nonlinear estimation and control. This structure proved to be flexible and robust.

Implementation of this neural network adaptive control structure was demonstrated on the system of longitudinal motion of the A-4 aircraft. Estimation and control capabilities were shown. First, the stability of a linear neural network estimator was demonstrated. Following this, two linear neural network estimators with different parameterizations were illustrated. The first, parameterized as four separate transfer functions, developed a fair model of the system while demonstrating the susceptibility of neural networks to a variety of problems known from estimation theory. The other, fully parameterized, neural network estimator modelled the system exactly. The similarities between estimation theory and neural networks was demonstrated using these two linear estimators.

Two nonlinear neural network estimators were then demonstrated. Very little theory exists to help determine the structure of nonlinear neural networks. For this investigation, empirically determined neural network estimation structures were used to develop models for linear and nonlinear systems. The linear system which was to be modelled was the same one used for the linear estimators. The nonlinear neural network which was used to model a linear system performed well, but not as good as the fully parameterized linear estimator. The nonlinear system to be modelled was formed by presenting a number of different linear models to the neural network. The nonlinear network which was used to model a nonlinear system was relatively successful at modelling the nonlinear flight conditions and generalizing for flight conditions on which it was not trained. These two demonstrations illustrated the capabilities of neural network estimators to model nonlinear systems.

Using the linear neural network adaptive control structure, noise rejection capabilities similar to those of other forms of adaptive controller were demonstrated. The neural network adaptive controller was highly successful at rejecting a random binary, white noise input.

This thesis has shown that neural networks have tremendous potential in the field of adaptive control. Further study on this specific adaptive control structure should be made.

The use of this structure to develop various combinations of nonlinear and linear control and estimation should be studied. For nonlinear networks, a theoretical basis for the number of elements, and convergence and stability characteristics are needed. For all types of neural network, better ways to adapt the gain need to be developed to avoid the problems of scaling and changing the learning rate. With the demands on current forms of control steadily increasing, the need for real time parallel distributed processing applications in control will become essential.

REFERENCES

1. Bavarian, Behnam, "Introduction to Neural Networks for Intelligent Control", *IEEE Control Systems Magazine*, v. 8, no. 2, pp. 3-7, April, 1988.

2. Rumelhart, D. E., J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, The MIT Press, 1988.

3. Klimasauskas, C.,and others, *Neuralworks Professional II Manual*, Neuralware, Inc., 1988.

4. Hecht-Nielsen, "Kolmogorov's Mapping Neural Network Existence Theorem", *IEEE First International Conference on Neural Networks, 1987*, The Institute of Electrical and Electronic Engineers, Inc., 1987.

5. Goodwin, G. C., and K. S. Sin, *Adaptive Filtering, Prediction, and Control*, Prentice-Hall, Inc., 1984.

6. Ljung, L., *System Identification: Theory for the User*, Prentice-Hall, Inc., 1987.

7. Astrom, K. J., and B. Wittenmark, *Adaptive Control*, p. 14, Addison-Wesley Publishing Co., 1989.

8. Kawato, M., and others, "Hierarchical Neural Network Model for Voluntary Movement with Application to Robotics", *IEEE Control Systems Magazine*, v. 8, no. 2, April 1988.

9. Sun Microsystems, Inc., *The Sun 386i. The Corporate 386*, 1988.

10. Neuralware, Inc., *Neuralworks Professional II on Sun Workstations*, 1989.

11. Moler, C., J. Little, and S. Bayert, *Pro-MATLAB User's Manual*, The Mathworks, Inc., 1897.

12. Etkin, B., *Dynamics of Flight--Stability and Control*, 2d ed., John Wiley & Sons, Inc., 1987.

13. Nelson, R. C., *Flight Stability and Automatic Control*, McGraw-Hill, Inc., 1989.

14. Harris, C. J., and S. A. Billings, ed., *Self-Tuning and Adaptive Control: Theory and Applications*, pp. 109-141, Institution of Electrical Engineers, 1981.

APPENDIX A: NEURALWORKS PROFESSIONAL II ASSOCIATED PROGRAMS

```
/*****************************************************************************
 * Source:      simo.txt
 * Executable: simo
 * Version:     3.1
 * Date:        22 November 1989
 * Author:      R. W. Scott
 * Project:     Neural Networks in Adaptive Control
 * Environment: UNIX/SunOS C
 * Path:        eileen:/home/rscott/nworks/textfiles
 * Description:This is a prototype for the USERIO program spawned by
 *              NWORKS Professional II to provide input and output
 *              vectors for the use of an adaptive control neural
 *              network.  The program operates by running a simulation
 *              of the longitudinal motion of the A-4 aircraft at
 *              the same speed as sampling time of the network.
 *              Numerous different input types are available.  The
 *              simulation may be run at various flight conditions as well.
 *              The structure for control is available, however, only the
 *              estimation portion of the program is provided.
 * Revisions:   --Inclusion of multiple input types
 *              --Inclusion of easy overparamaterization
 *****************************************************************************/

/* Include the following external modules */

#include <stdio.h>
#include <math.h>
#include "userutl.h"
#include "transfer.h"                   /* File of parameters */

/* Neuralworks calls the USERIO program through the function UsrIO */

int UsrIO()
        {

/* Declarations */

        extern double sin();            /* Sine function */
        extern double pow();            /* Power function */
        extern double fmod();           /* Remainder function */
        extern long random();           /* Random number generator */
        extern char *condition_name[];/* Names of conditions */
        extern char *input_name[];  /* Names of inputs */
        extern char *filter_name[];  /* Names for filters */
        extern double altitude[];       /* Altitudes */
        extern double mach[];           /* Mach numbers */
        extern double noise_coeff[3][5];
        extern double num[5][4][4]; /* Numerator coefficients */
        extern double den[5][4];        /* Denominator coefficients */
        extern double freq[];           /* Frequencies for composite sine */
        extern double weights[];        /* Weighting of frequencies */
        extern double ts;               /* Sampling time */
```

```
        /* Random phase for sine waves */
        static double phase[8]={0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};

        static int profile={0};
        static  double tot_wt={0.0};/* Total weight for comp sinc */
        static int redraw,in={0};    /* Redisplay initialization flag */
        static double check1;        /* Check flag */
        static double check2;        /* Check flag */
        static double count={0.0};   /* Display counter */
        static int condition;        /* Selected condition */
        static int input;            /* Selected input */
        static int filter;           /* Selected filter */

        /* RBS Uniformly Distributed White Noise Sequence */
        static double noise[5]={0.0,0.0,0.0,0.0,0.0};



        /* Fedback regression vector */
        static double feedback[19]={0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
                0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};

        /* Ref input + regression vec */
        static double command[20]={0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
                0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};

        /* Regression vector applied to NN */
        static double control[20]={0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
                0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};

        /* Plant response to regression vector */
        static double plant[4]={0.0,0.0,0.0,0.0};

        /* Model reference output */
        static double reference[4]={0.0,0.0,0.0,0.0};

        static double rcount,rmem;  /* Counters for generating inputs */
                int i,j;            /* Indices */
                char buf[90];       /* Display buffer */
                char *sp;           /* String pointer */

/* Definitions */

#define MAXRAND (0x7fffffff1)
#define rand random

/* Define layer names */

#define feedback_lay 0
#define command_lay 1
#define control_lay 2
#define plant_lay 3
#define reference_lay 4

/* initialization here (if necessary) */

IORTNCDE = 0;
```

```
        switch ( IOREQCDE ) {

        case RQ_ATTENTION:

/* User select input to be used */

        Again3:
        sprintf( buf,"\nEnter Desired Input Type (1. %s, 2. %s, 3. %s",
                input_name[1],input_name[2].input_name[3]);
        PutStr( buf );
        sprintf(buf,"\n4. %s, 5. %s, or 6. %s)",
                input_name[4],input_name[5].input_name[6]);
        PutStr( buf);
        sp=GetStr();
        sscanf( sp, "%ld", &input);
        if( input >6. || input<1. ){
                sprintf(  buf, "\n%s",input_name[0] );
                PutStr( buf );
                for(i=0;i<1000;i++){
                }
                goto Again3;
        }

/* User select condition to be used */

        Again:
        sprintf( buf,"\nEnter Desired Mach Number and Altitude (1. %s, 2. %s,",
                condition_name[1],condition_name[2]);
        PutStr( buf );
        sprintf( buf,"\n 3. %s, 4. %s, or 5. %s)",
                condition_name[3],condition_name[4],condition_name[5] );
        PutStr( buf );
        sp=GetStr();
        sscanf( sp, "%ld", &condition);
        if( condition >5. || condition<1. ){
                sprintf(  buf, "\n%s",condition_name[0] );
                PutStr( buf );
                for(i=0;i<1000;i++){
                }
                goto Again;
        }
        if(input==2||input==5||input==6){
        Again2:
        sprintf( buf,"\nEnter Desired Filter (1. %s, 2. %s, or 3. %s)?",
                filter_name[1],filter_name[2].filter_name[3]);
        PutStr( buf );
        sp=GetStr();
        sscanf( sp, "%ld", &filter);
        if( filter >3. || filter<1. ){
                sprintf(  buf, "\n%s",filter_name[0] );
                PutStr( buf );
                for(i=0;i<1000;i++){
                }
        goto Again2;
        }
        }

/* Display selections */

        sprintf( buf,"\nCondition: %s Input: %s selected",
                condition_name[condition],input_name[input]);
        PutStr( buf );
        if(input==4){
                PutStr("\nEnsure LR is set to zero for test");
        }
        in=1;
        break;
```

```
        case RQ_REWIND:

/* rewind here */

        count = 0.0;
        break;

        case RQ_LSTART:

/* learn start */


/* initialize condition and input if not already done so */

        if(in = = 0){
                input = 1;
                condition = 1;
                in = 1;
        }

/* check if user wishes to redisplay after every plot reaches the end */

        PutStr("\nHow often do you wish to redraw the screen (0 for never)?");
        sp = GetStr();
        sscanf( sp, "%ld", &redraw);

/* compute the total weights for composite sine wave */

        for(i = 1;i < 8;i + +){
                tot_wt = weight[i] + tot_wt;
        }

/* start random binary or composite in time sequence */

        if(input = = 1 || input = = 6){
                rcount = 0.0;
                rmem = rand() % 4;
                command[0] = pow(-1.0,rmem):
        }

/* start filtered random binary sequence */




        if(input = = 2){
                rmem = rand() % 4;
                noise[0] = pow(-1.0,rmem);
                command[0] = noise_coeff[filter-1][0]*noise[0];
                for(i = 1;i < 3;i + +){
                command[0] = command[0] + noise_coeff[filter-1][i]*noise[i];
                command[0] = command[0]-noise_coeff[filter-1][i + 2]*noise[i + 2];
                noise[i] = noise[i-1];
                }
                        noise[4] = noise[3]:
                        noise[3] = command[0];
        }
```

```
/* start composite sine wave sequence */

        if(input==3){
                for(i=0;i<8;i++){
                        rmem=rand() % 10;
                        phase[i]=rmem*3.1415927/5.0;
                }
                command[0]=0.0;
                for(i=0;i<8;i++){
command[0]=command[0]+weight[i]/tot_wt*sin((freq[i])*count*ts+(phase[i]));
                }
        }

/* start test sequence (swept square wave) */

        if(input==4){
                rmem=90.0;
                rcount=90.0;
                command[0]=pow(-1.0,rmem);
        }

/* start composite simultaneous sequence */

        if(input==5){
                rmem=rand() % 4;
                command[0]=0.1*pow(-1.0,rmem);
                rmem=rand() % 4;
                noise[0]=pow(-1.0,rmem);
                command[0]=command[0]+noise_coeff[filter-1][0]*noise[0];
                for(i=1;i<3;i++){
                command[0]=command[0]+noise_coeff[filter-1][i]*noise[i];
                command[0]=command[0]-noise_coeff[filter-1][i+2]*command[i];
                        noise[i]=noise[i-1];
                }
        }


/* display the starting conditions */

        sprintf( buf, "\nCondition: %s Cycles: %f Input: %s",
                condition_name[condition], count, input_name[input]);
        PutStr( buf );
        break;

        case RQ_LEARNIN:

/* input command layer to the network */

        if( IOLAYER==feedback_lay && IOCOUNT==19 ){
                for( i=0; i<19;i++ ){
                        IODATA[i]=feedback[i];

                }
        }

/* input feedback layer to the network */

        if( IOLAYER==command_lay && IOCOUNT==20 ){
                for(i=0;i<20;i++){
                        IODATA[i]=command[i];
                }
        }

        break;
```

```
        case RQ_WRSTEP:

/* output control layer from network */

        break;

        case RQ_LEARNOUT:

/* present plant or model response to the network */

        if( IOLAYER==plant_lay && IOCOUNT== 4){
                for(i=0;i<4;i++){
                IODATA[i]=plant[i];
                }
        }

        if( IOLAYER==reference_lay && IOCOUNT==4){
                for(i=0;i<4;i++){
                IODATA[i]=reference[i];
                }
        }

        break;

        case RQ_LEARNRSLT:

/* control output from network */

        if( IOLAYER==control_lay && IOCOUNT==1 ){

                for(i=0;i<20;i++){
                        control[i]=command[i];
                }
                control[0]=IODATA[0];
/* generate system and model response to this control input */

                for(i=0;i<4;i++){
                        plant[i]=0.00;
                        reference[i]=0.00;
                        for(j=0;j<4;j++){
                plant[i]=plant[i]+num[(condition-1)][i][j]*(control[j]);
                plant[i]=plant[i]+den[(condition-1)][j]*(control[4*(i+1)+j]);
                        }
                reference[i]=reference[i]+num[1][i][0]*command[0];
                reference[i]=reference[i]+den[1][0]*control[4*(i+1)];
                for(j=1;j<4;j++){
                reference[i]=reference[i]+num[1][i][j]*control[j];
                reference[i]=reference[i]+den[1][j]*control[4*(i+1)+j];
                        }
                }
                for(i=0;i<4;i++){
                        if(plant[i]>100.0 || plant[i]<-100.0){
                                plant[i]=0.0;
                        }
                        if(reference[i]>100.000 || reference[i]<-100.0){
                                plant[i]=0.0;
                        }
                }
        }
```

```
/* system identification result out from network */
        if(IOCOUNT==4 && IOLAYER==plant_lay){

/* shift the regression vectors */

                for(i=0;i<19;i++){
                        feedback[i]=control[i];
                }
                for (i=0;i<19;i++){
                        command[i+1]=control[i];
                }

/* generate a new random binary input */

                if(input==1){
                        rcount++;
                        command[0]=command[1];
                        if(fmod(count,2.0)<1.0){
                                rmem=rand() % 4;
                                command[0]=pow(-1.0,rmem);
                        }
                }
/* generate a new filtered random binary input */

                if(input==2){
                        rmem=rand() % 4;
                                noise[0]=pow(-1.0,rmem);
                                command[0]=noise_coeff[filter-1][0]*noise[0];
                                for(i=1;i<3;i++){
                command[0]=command[0]+noise_coeff[filter-1][i]*noise[i];
                command[0]=command[0]-noise_coeff[filter-1][i+2]*noise[i+2];
                noise[i]=noise[i-1];
                                }
                        noise[4]=noise[3];
                        noise[3]=command[0];

                }

/* generate a new composite sine wave input */

                if(input==3){
                        command[0]=0;
                        for(i=0;i<8;i++){
      command[0]=command[0]+weight[i]/tot_wt*sin((freq[i])*count*ts+(phase[i]));
                        }
                }

/* generate a new composite simultaneous input */

                if(input==5){
                rmem=rand() % 4;
                noise[0]=pow(-1.0,rmem);
                command[0]=noise_coeff[filter-1][0]*noise[0];
                for(i=1;i<3;i++){
                command[0]=command[0]+noise_coeff[filter-1][i]*noise[i];
                command[0]=command[0]-noise_coeff[filter-1][i+2]*noise[i+2];
                        noise[i]=noise[i-1];
                }
                noise[4]=noise[3];
                noise[3]=command[0];
                rmem=rand() % 4;
                command[0]=command[0]+0.1*pow(-1.0,rmem);
                }
```

```
/* generate a new test (swept square wave) input */

        if(input = =4){
                rcount--;
                if (rcount< =0.0){
                        rmem--;
                        rcount = rmem;
                        if (rmem< =0.0){
                                rmem = 90.0;
                                rcount = 90.0;
                        }
                }
                command[0] = pow(-1.0,rmem);
        }

/* load the regressors with system and model responses */

        for(i = 0;i < 4;i + +){
                command[4*(i + 1)] = -plant[i];
                feedback[4*(i + 1)-1] = -plant[i];
        }

/* increment the counter and update displays as necessary */

        count + +;
        check1 = fmod(count,10.);

        if(check1 < 1.0){
        sprintf( buf, "\nCondition: %s Cycles: %f Input: %s",
                condition_name[condition], count,input_name[input]);
                PutStr( buf );
        }
        check2 = fmod(count,300.);
        if(check2 < 1.0){
                profile + +;
                        if (profile > =30){
                                for (i = 0;i < 19;i + +){
                                feedback[i] = 0.0;
                                command[i + 1] = 0.0;
                                }
                                profile = 0;
                        }
        }
        if(redraw != 0){
                if(fmod(count,(double)redraw) < 1.0){
                        IORTNCDE = 1;
                }
        }
```

```
/* generate a new composite in time input (requires the use of counters) */

            if(input==6){
                if(check1>=check2){
                        rmem=rand() % 4;
                        command[0]=pow(-1.0,rmem);
                        noise[3]=0.0;
                        noise[4]=0.0;
                }
                else{
                        rmem=rand() % 4;
                        noise[0]=pow(-1.0,rmem);
                        command[0]=noise_coeff[filter-1][0]*noise[0];
                        for(i=1;i<3;i++){
            command[0]=command[0]+noise_coeff[filter-1][i]*noise[i];
            command[0]=command[0]-noise_coeff[filter-1][i+2]*noise[i+2];
            noise[i]=noise[i-1];
                        }
                noise[4]=noise[3];
                noise[3]=command[0];
                }
            }

        }

/* control result out from network */

        if(IOCOUNT==4 && IOLAYER==reference_lay){
                }
        break;

        case RQ_LEND:

/* end learning mode, display current status */

        sprintf( buf, "\nCondition: %s Cycles: %f Input: %s",
                condition_name[condition], count, input_name[input]);
                PutStr( buf );

        break;


        case RQ_RSTART:


        break;

        case RQ_READ:


        break;


        case RQ_WRITE:

        break;

        case RQ_REND:

        /* end recall */


        break;
```

```
        case RQ_TERM:

        /* terminate userio */

        sprintf( buf, "\nCondition:  %s  Cycles:  %f",
                      condition_name[condition],  count);
                      PutStr( buf );

        break;

        }

        return;
}
```

```
/*************************************************************************
 * Source:      transfer.txt
 * Executable: simo
 * Version:     1.5
 * Date:        22 November 1989
 * Author:      R. W. Scott
 * Project:     Neural Networks in Adaptive Control
 * Environment:   UNIX/SunOS C
 * Path:        eileen:/home/rscott/nworks/textfiles
 * Description:This is the header file used to define the variables
 *              used in the USERIO subprogram simo.  This allows easy
 *              reconfiguration of the executables by simply changing
 *              information in the header file.  Inputs include altitudes,
 *              airspeeds, the sampling time, selected frequencies and
 *              and weightings for a sum of sine waves input, labels for the
 *              inputs, conditions, and states, and the coefficients for the
 *              numerators and denominators of the system and various filters
 *              used to generate filtered noise.
 * Revisions:   --Inclusion of multiple input types
 *************************************************************************/


/* Altitudes in thousands of feet */

static double altitude[5]=
        {
        0.0,0.150,0.350,0.0,0.350
        };

/* Mach Numbers */

static double mach[5]=
        {
        0.4,0.5,0.6,0.8,0.8
        };

/* Sampling Time */

static double ts={0.1};

/* Frequencies for sum of sine waves input */

static double frcq[8]={
                0.005,0.09,0.11,0.65,1.5,2.75,3.0,10.0
                };

/* Frequency weighting for sum of sine waves input */

static double weight[8]={
                2.0,3.0,2.0,3.0,2.0,3.0,2.0,0.5
                };

/*  Input, condition, state, and filter labels */

static char *input_name[]={"Illegal Input","Random Binary",
        "Filtered RB","Composite Sine","Swept Square Wave-Test Only",
        "Composite Sim","Composite Time"};

static char *condition_name[]={"Illegal Condition",
        "M 0.4/SL", "M 0.5/15K", "M 0.6/35K", "M 0.8/SL", "M 0.8/35K"};

static char *state_name[]={"Illegal State","u(t)","alpha(t)",
        "q(t)", "theta(t)"};

static char *filter_name[]={"Illegal Filter","0.5 Hz co",
        "0.2 Hz co","Alpha App M 0.5/15K"};
```

```
/* Numerator coefficients */
/* Order is u1-u4,a1-a4,q1-q4,t1-t4 for the inner indices and
   Condition 1-Condition 5 for the outer index */
static double num[5][4][4] =
        {       1.248543503451494e-04,
     3.554904582170337e-04,
    -3.247047792447333e-04,
    -1.039094082966319e-04,
    -6.756262084831643e-02,
     8.786514170600235e-02,
     2.676626371037605e-02,
    -4.708037372835860e-02,
    -1.173118751144243e+00,
     3.425397067901477e+00,
    -3.331568421196559e+00,
     1.079290104439327e+00,
    -6.062231436411736e-02,
     5.883590144983231e-02,
     5.479098533816495e-02,
    -5.301772049566766e-02,
     1.733432397501566e-04,
     4.664138711447663e-04,
    -4.606405022249405e-04,
    -1.388039811326403e-04,
    -6.509790082128220e-02,
     8.131467412035409e-02,
     3.246072975226699e-02,
    -4.868611382480093e-02,
    -1.159429186560191e+00,
     3.404538807812190e+00,
    -3.330869731837395e+00,
     1.085760110585395e+00,
    -5.953704958760442e-02,
     5.808824962831594e-02,
     5.505421438561653e-02,
    -5.361344543414581e-02,
     1.683431989607520e-04,
     4.427763960341835e-04,
    -4.616352316690886e-04,
    -1.351063663521668e-04,
    -4.275127449734661e-02,
     5.171905532394572e-02,
     2.472773164204334e-02,
    -3.369874470285861e-02,
    -7.719049116594103e-01,
     2.289294061271993e+00,
    -2.262864984231793e+00,
     7.454758346192119e-01,
    -3.919528344719048e-02,
     3.872425877625929e-02,
     3.749529462200130e-02,
    -3.702344394883295e-02,
     6.677957396172829e-05,
     2.425576303810573e-04,
    -1.782118437363422e-04,
    -7.369054508732376e-05,
    -9.527558246644441e-02,
     1.115637753090919e-01,
     6.244296581944386e-02,
    -7.873812158122262e-02,
    -1.767708225383969e+00,
     5.198238987741318e+00,
    -5.093536378204770e+00,
     1.663005615847422e+00,
    -9.111095856528628e-02,
     8.934216069297385e-02,
     8.414788921865490e-02,
    -8.239747524425589e-02,
     6.677957396172829e-05,
     2.425576303810573e-04,
```

```
      -1.782118437363422e-04,
      -7.369054508732376e-05,
      -9.527558246644441e-02,
       1.115637753090919e-01,
       6.244296581944386e-02,
      -7.873812158122262e-02,
      -1.767708225383969e+00,
       5.198238987741318e+00,
      -5.093536378204770e+00,
       1.663005615847422e+00,
      -9.111095856528628e-02,
       8.934216069297385e-02,
       8.414788921865490e-02,
      -8.239747524425589e-02
                };

/* Denominator coefficients */
/* Order is den1-den4 for the inner index and Condition 1-Condition 5
   for the outer index */
static double den[5][4]=
         {
          -3.694923643854825e+00,
       5.180217304754835e+00,
      -3.275499735648207e+00,
       7.902148612567820e-01,
      -3.746857513326851e+00,
       5.326510948543416e+00,
      -3.412294750223168e+00,
       8.326474112728178e-01,
      -3.851514509136133e+00,
       5.606710127567315e+00,
      -3.658817410841104e+00,
       9.036239272554680e-01,
      -3.712165762093378e+00,
       5.260223167365153e+00,
      -3.383762647629577e+00,
       8.357099953933538e-01,
      -3.759751975887927e+00,
       5.374549739965524e+00,
      -3.469699533531960e+00,
       8.549056369244077e-01
         };

/* Coeffieients for filtered noise terms */
/* Order is num1-num3 & den1-den2 for the inner index and filter1-
   filter3 for the outer index */

static double noise_coeff[3][5]=
         {
         2.085670251279634e-02,
         4.171340502559269e-02,
         2.085670251279634e-02,
         -1.561018075800718e+00,
         6.413515380575631e-01,
         5.063654276859733e-03 ,
         1.012730855371947e-02,
         5.063654276859733e-03,
         -1.822694925196308e+00,
         8.371816512560227e-01,
         0.0,
         -3.335605497273741e-02,
         -2.498849406787340e-02,
         -1.748500141242948e+00,
         8.340433823724368e-01
         };
```

```
csv2.1
!file format is Control Strategy Version 2.1
!
!
!  Source:       hidden0.nnc
!  Executable:   neuralworks professional II
!  Version:      1.3
!  Date:         22 November 1989
!  Author:       R. W. Scott
!  Project:      Neural Networks in Adaptive Control
!  Environment:    UNIX/SunOS/Neuralworks Control Strategy
!  Path:         eileen:/home/rscott/nworks/textfiles
!  Description: This is a prototype control strategy for use with
!               and the simo USERIO program.  The recall strategy is not
!               used.  The control and identification strategies determine
!               sequence in which propagation and learning take place as
!               as the manner in which layers are altered.
!               This strategy uses a proprietary language which is covered
!                  in some detail in the Neuralworks Professional II manual.
!  Revisions:   No major revisions
!
!
!MASK  label op-code       operandscomment
L_saR_sa      optclr op:bknc       !   do not BKp to PEs w/o conns
L_saR_sa      trace  aux3          !   set trace option to aux3
Li_aR_sa      cset   recall,0!   recall count
!
! Recall Strategy
!
L__R_sa       lset   in            !   set command layer
L__R_sa       io     read          !   get command vector
L__R_sa       lset   cur,1         !   set feedback layer
L__R_sa       io     read          !   get feedback vector
L__R_sa       lset   in            !   set command layer
L__R_sa       math   sum¦rnoise¦tran¦output¦e=0  !   fire 1st layer
L__R_sa       lset   cur,1         !   set feedback layer
L__R_sa       math   sum¦rnoise¦tran¦output¦e=0  !   fire 2nd layer
L__R_sa       lset   cur,1         !   set control layer
L__R_sa       math   sum¦rnoise¦tran¦output¦e=0  !   fire 3rd layer
L__R_sa       lset   cur,1         !   set plant layer
L__R_sa       math   sum¦rnoise¦tran¦output¦e=0  !   fire 4th layer
L__R_sa       lset   cur,1         !   set reference layer
L__R_sa       math   sum¦rnoise¦tran¦output  !   fire 5th layer
L__R_sa       io     write         !   write recall result to userio
!
! Control Strategy
L__aR___      ccmp   epoch,aux1    !  test for end of sys-id
L__aR___      blt    @id           !  branch to id phase
L__aR___      lset   in            !   set command layer
L__aR___      io     lrnin         !   get command vector
L__aR___      lset   cur,1         !   set feedback layer
L__aR___      io     lrnin         !   get feedback vector
L__aR___      lset   out           !   set reference layer
L__aR___      io     lrnout        !   get reference vector
L__aR___      lset   in            !   set command layer
L__aR___      math   sum¦lnoise¦tran¦output¦e=0¦fire  !   fire 1st layer
L__aR___      lset   cur,1         !   set feedback layer
L__aR___      math   sum¦lnoise¦tran¦output¦e=0¦fire  !   fire 2nd layer
L__aR___      lset   cur,1         !   set control layer
L__aR___      math   sum¦lnoise¦tran¦output¦e=0¦fire  !   fire 3rd layer
L__aR___      io     lrnrslt       !   write control result to userio
L__aR___      lset   cur,1         !   set plant layer
L__aR___      math   sum¦lnoise¦tran¦output¦e=0¦fire  !   fire 4th layer
L__aR___      lset   cur,1         !   set reference layer
L__aR___      math   sum¦lnoise¦tran¦output¦e-=w¦fire !  fire 5th
L__aR___      lset   out           !   set reference layer
L__aR___      math   ee=e¦e*=f'¦backp¦fire  !   bkp 5th layer
L__aR___      lset   cur,-1        !  set plant layer
L__aR___      math   ee=e¦e*=f'¦backp¦fire  !   bkp 4th layer
L__aR___      lset   cur,-1        !   set control layer
```

```
L_aR___      math  ce=e¦e*=f'¦backp¦learn¦fire  !   bkp 3rd layer
L_aR___      lset  cur,-1          !   set control layer
L_aR___      math  ce=e¦e*=f'¦backp¦learn¦fire  !   bkp 3rd layer
!
! System Identification Strategy
!
L_aR___  @id      lset  in          !   set command layer
L_aR___      io    lrnin           !   get command vector
L_aR___      lset  cur,1           !   set feedback layer
L_aR___      io    lrnin           !   get feedback vector
L_aR___      lset  in              !   set command layer
L_aR___      math  sum¦lnoise¦tran¦output¦e=0¦fire  !   fire 1st layer
L_aR___      lset  cur,1           !   set feedback layer
L_aR___      math  sum¦lnoise¦tran¦output¦e=0¦fire  !   fire 2nd layer
L_aR___      lset  cur,1           !   set control layer
L_aR___      math  sum¦lnoise¦tran¦output¦e=0¦fire  !   fire 3rd layer
L_aR___      io    lrnrslt         !  send control inputs to userio
L_aR___      lset  cur,1           !   set plant layer
L_aR___      io    lrnout          !   get plant vector from userio
L_aR___      math  sum¦lnoise¦tran¦output¦e-=w¦fire !  fire 4th
L_aR___      io    lrnrslt         !   write sys id result to userio
L_aR___      lset  out             !   set reference layer
L_aR___      lset  cur,-1          !  set plant layer
L_aR___      math  ce=e¦e*=f'¦backp¦learn¦fire  !   bkp/learn 4th
L_aR_sa      trace 0               !   turn off any trace function
```

APPENDIX B: MATLAB M-FILE

```
% css2dtf.m
% Continuous state space to discrete transfer function conversion.
% Required inputs:
%     system continuous a & b matrices
%     t - sampling time
%
% Outputs:
%     ab, bb, cb balanced state space
%     ad, bd     discrete matrices
%     ns, ds          numerator and denominator of discrete transfer function
%
% Convert b from radians to degrees
bx=b*pi/180;
%
% Scale outputs
c=[.02172986525780895,29.358526682723,9.7019323535787,7.282373233084900];
c=diag(c);
d=zeros(4,1);
%
% Balance a, b, and c matrices
[ab,bb,cb]=obalreal(a,bx,c);
%
% Convert to discrete time
[ad,bd]=c2d(ab,bb,t);
%
% Convert jto transfer function
[ns,ds]=ss2tf(ad,bd,cb,d,1);
```

Sampling Time of 0.1 Seconds

Flight Condition 1

a =

```
 -1.5154e-02  -2.2559e+00         0  -3.2174e+01
 -3.1672e-04  -8.7896e-01   1.0000e+00         0
  1.0825e-04  -9.4643e+00  -1.4604e+00         0
         0          0   1.0000e+00          0
```

b =

```
         0
 -9.0927e-02
 -1.2848e+01
         0
```

alt =

```
    0
```

u =

```
  4.4658e+02
```

mach =

```
  4.0000e-01
```

ns =

```
         0   2.7131e-05   7.7248e-05  -7.0558e-05  -2.2579e-05
         0  -3.4619e-02   4.5022e-02   1.3715e-02  -2.4124e-02
         0  -1.9864e-01   5.8002e-01  -5.6414e-01   1.8276e-01
         0  -7.7052e-03   7.4781e-03   6.9640e-03  -6.7386e-03
```

ds =

```
  1.0000e+00  -3.6949e+00   5.1802e+00  -3.2755e+00   7.9021e-01
```

Flight Condition 2

a =

-1.6751e-02	-1.4926e+01	0	-3.2174e+01
-2.3639e-04	-6.8474e-01	1.0000e+00	0
6.4590e-05	-8.6540e+00	-1.1300e+00	0
0	0	1.0000e+00	0

b =

0
-7.1523e-02
-1.2468e+01
0

alt =

15000

u =

5.2868e+02

mach =

5.0000e-01

ns =

0	3.7667e-05	1.0135e-04	-1.0010e-04	-3.0162e-05
0	-3.3356e-02	4.1666e-02	1.6633e-02	-2.4947e-02
0	-1.9633e-01	5.7649e-01	-5.6402e-01	1.8385e-01
0	-7.5672e-03	7.3831e-03	6.9975e-03	-6.8143e-03

ds =

1.0000e+00	-3.7469e+00	5.3265e+00	-3.4123e+00	8.3265e-01

Flight Condition 3

a =

```
 -1.0871e-02  -3.5930e+01            0  -3.2174e+01
 -1.3174e-04  -3.6286e-01   1.0000e+00            0
  2.0777e-05  -5.2634e+00  -6.3969e-01            0
           0            0   1.0000e+00            0
```

b =

```
           0
  -4.1024e-02
  -8.0491e+00
           0
```

alt =

```
       35000
```

u =

```
   5.8388e+02
```

mach =

```
   6.0000e-01
```

ns =

```
           0   3.6581e-05   9.6215e-05  -1.0031e-04  -2.9358e-05
           0  -2.1906e-02   2.6501e-02   1.2671e-02  -1.7267e-02
           0  -1.3071e-01   3.8765e-01  -3.8317e-01   1.2623e-01
           0  -4.9818e-03   4.9219e-03   4.7657e-03  -4.7057e-03
```

ds =

```
   1.0000e+00  -3.8515e+00   5.6067e+00  -3.6588e+00   9.0362e-01
```

Flight Condition 4

a =

-1.4120e-02	1.8216e+01	0	-3.2174e+01
-1.2471e-04	-6.3225e-01	1.0000e+00	0
3.9715e-05	-1.2887e+01	-1.1484e+00	0
0	0	1.0000e+00	0

b =

0
-6.2754e-02
-1.9154e+01
0

alt =

0

u =

8.9316e+02

mach =

8.0000e-01

ns =

0	1.4511e-05	5.2707e-05	-3.8725e-05	-1.6013e-05
0	-4.8819e-02	5.7166e-02	3.1996e-02	-4.0346e-02
0	-2.9933e-01	8.8022e-01	-8.6249e-01	2.8160e-01
0	-1.1580e-02	1.1356e-02	1.0695e-02	-1.0473e-02

ds =

1.0000e+00	-3.7122e+00	5.2602e+00	-3.3838e+00	8.3571e-01

Flight Condition 5

a =

-1.4495e-02	-2.7679e+01	0	-3.2174e+01
-1.3174e-04	-5.5219e-01	1.0000e+00	0
3.6568e-05	-9.7906e+00	-1.0010e+00	0
0	0	1.0000e+00	0

b =

0
-5.4699e-02
-1.4552e+01
0

alt =

35000

u =

7.7851e+02

mach =

8.0000e-01

ns =

0	5.5856e-05	1.5033e-04	-1.5189e-04	-4.5898e-05
0	-3.7808e-02	4.4639e-02	2.4041e-02	-3.0875e-02
0	-2.3030e-01	6.7891e-01	-6.6693e-01	2.1832e-01
0	-8.8646e-03	8.7106e-03	8.2723e-03	-8.1190e-03

ds =

1.0000e+00	-3.7598e+00	5.3745e+00	-3.4697e+00	8.5491e-01

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chairman, Code 67 Department of Aeronautics and Astronautics Naval Postgraduate School Monterey, California 93943-5002	1
4.	Professor D. J. Collins Code 67Co Department of Aeronautics and Astronautics Naval Postgraduate School Monterey, California 93943-5002	5
5.	Professor J. P. Hauser Code 67Ha Department of Aeronautics and Astronautics Naval Postgraduate School Monterey, California 93943-5002	1
6.	Professor J. Burl Code 62Bl Department of Electrical Engineering Naval Postgraduate School Monterey, California 93943-5002	1
7.	LCDR Roger Stemp Code 30 Operations Analysis Curricular Officer Naval Postgraduate School Monterey, California 93943-5002	1
8.	Mr. Tor Jensen Code 6013 Naval Air Development Center Warminster, Pennsylvania 18974	1
9.	Mr. Joe Gera NASA Dryden Flight Research Center P. O. Box 273 Mail Code OFDC Edwards, California 93523	1
10.	Mr. Thomas Momiyama AIR 931 Naval Air Systems Command Washington, D.C. 20361-0001	1

11. Mr. George Derderian 1
AIR 931E
Naval Air Systems Command
Washington, D.C. 20361-0001

12. LT Russell W. Scott 2
AIR 54661C
Naval Air Systems Command
Washington, D. C. 20361-0001